

# Table of Contents

[Build and Release](#)

[Overview](#)

[About Build and Release](#)

[About Deploying to Azure](#)

[Quickstarts](#)

[Azure DevOps projects](#)

[ASP.NET Core](#)

[Node.js](#)

[Java](#)

[Python](#)

[PHP](#)

[Build your app](#)

[Android](#)

[ASP.NET Core](#)

[ASP.NET](#)

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

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

[Docker image](#)

[Go](#)

[Gradle](#)

[Maven](#)

[.NET for Windows](#)

[Node.js](#)

[Universal Windows Platform](#)

[Xamarin](#)

[Xcode](#)

[Deploy your app](#)

[Azure Web App](#)

[Windows VM](#)

- [Linux VM](#)
- [Web App for Containers](#)
- [Test your app](#)
- [VS Test](#)
- [Tutorials](#)
  - [CI builds for Git in VSTS](#)
  - [CI builds for GitHub repo](#)
  - [Set up multi-stage release](#)
  - [Use approvals and gates](#)
  - [Azure DevOps Project for GitHub](#)
- [Concepts](#)
  - [Agents](#)
    - [Build and release agents](#)
    - [Agent pools and queues](#)
    - [Hosted agents](#)
    - [Deployment groups](#)
  - [Builds](#)
    - [Artifacts](#)
    - [History](#)
    - [Options](#)
    - [Repository](#)
    - [Triggers](#)
    - [Variables](#)
  - [Releases](#)
    - [What is Release Management?](#)
    - [Release definitions](#)
    - [Environments](#)
    - [Artifacts](#)
    - [Triggers](#)
    - [Variables](#)
    - [Approvals and gates](#)
    - [Templates](#)

## [Releases and deployments](#)

### [Process](#)

[Tasks](#)

[Phases](#)

[Conditions](#)

### [Licensing](#)

[Pipelines in TFS](#)

[Pipelines in VSTS](#)

### [Library](#)

[Variable groups](#)

[Task groups](#)

[Service endpoints](#)

[Secure files](#)

### [Retention policies](#)

## [How-to Guides](#)

### [Deploy an agent](#)

[Windows](#)

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

[macOS](#)

[Linux](#)

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

[Run an agent with self-signed certificate](#)

[Configure TFS authentication](#)

### [Use YAML CI \(config as code\)](#)

### [Build more app types](#)

[Azure Cloud Service](#)

[SQL Server Database](#)

[Run a PowerShell script](#)

[Run Git commands](#)

[Sign your mobile app](#)

### [Deploy to more targets](#)

[Azure web app](#)

- [Extend App Services deployments](#)
- [Extend IIS Server deployments](#)
- [Azure cloud service](#)
- [Azure Government Cloud](#)
- [Azure Container Service \(Kubernetes\)](#)
- [IIS servers \(WinRM\)](#)
- [Azure SQL database \(DACPAC\)](#)
- [Azure SQL database \(SQL scripts\)](#)
- [SCVMM](#)
- [SCVMM actions for managing VMs](#)
- [VMware](#)
- [Test your apps](#)
  - [Test as you build](#)
  - [Publish symbols for debugging](#)
    - [Publish symbols](#)
  - [Use packages in CI/CD](#)
    - [NuGet](#)
    - [npm](#)
    - [Maven](#)
  - [Migrate](#)
    - [Migrate from XAML builds](#)
    - [Migrate from Lab Management](#)
    - [Use Build & Release for automated testing](#)
    - [Set build and release permissions \(Security\)](#)
  - [Troubleshooting](#)
    - [Troubleshooting build](#)
    - [Debug deployment issues](#)
    - [Troubleshoot Azure connections](#)
  - [Reference](#)
    - [Build tasks](#)
      - [.NET Core](#)
      - [.NET Core CLI](#)

- [Android build](#)
- [Android signing](#)
- [Ant](#)
- [CMake](#)
- [Gradle](#)
- [Grunt](#)
- [Gulp](#)
- [Index Sources & Publish Symbols](#)
- [Jenkins Queue Job](#)
- [Maven](#)
- [MSBuild](#)
- [Visual Studio Build](#)
- [Xamarin.Android](#)
- [Xamarin.iOS](#)
- [Xcode](#)
- [Xcode Package iOS](#)
- [Utility tasks](#)
  - [Archive files](#)
  - [Azure Function](#)
  - [Azure Monitor](#)
  - [Batch script](#)
  - [Command line](#)
  - [Copy and Publish Build Artifacts](#)
  - [Copy Files](#)
  - [cURL Upload Files](#)
  - [Delay](#)
  - [Delete Files](#)
  - [Download Secure File](#)
  - [Extract Files](#)
  - [FTP Upload](#)
  - [Install Apple Certificate](#)
  - [Install Apple Provisioning Profile](#)

- [Invoke REST API](#)
- [Manual Intervention](#)
- [PowerShell](#)
- [Publish Build Artifacts](#)
- [Publish to Azure Service Bus](#)
- [Query Work Items](#)
- [Service Fabric PowerShell](#)
- [Shell script](#)
- [Update Service Fabric App Versions](#)
- [Xamarin license](#)
- [Test tasks](#)
  - [Cloud-based Apache JMeter Load Test](#)
  - [Cloud-based Load Test](#)
  - [Cloud-based Web Performance Test](#)
  - [App Center Test](#)
  - [Publish Code Coverage Results](#)
  - [Publish Test Results](#)
  - [Visual Studio Test](#)
  - [Xamarin Test Cloud](#)
- [Package tasks](#)
  - [CocoaPods](#)
  - [npm](#)
  - [NuGet](#)
  - [Xamarin Component Restore](#)
  - [Previous versions](#)
- [Deploy tasks](#)
  - [App Center Distribute](#)
  - [Azure App Service Deploy](#)
  - [Azure App Service Manage](#)
  - [Azure CLI](#)
  - [Azure Cloud Service Deployment](#)
  - [Azure File Copy](#)

[Azure Key Vault](#)

[Azure PowerShell](#)

[Azure Resource Group Deployment](#)

[Azure SQL Database Deployment](#)

[Copy Files Over SSH](#)

[IIS Web App Deploy](#)

[IIS Web App Manage](#)

[PowerShell on Target Machines](#)

[Service Fabric App Deployment](#)

[Service Fabric Compose Deploy](#)

[SSH](#)

[Windows Machine File Copy](#)

[Tool tasks](#)

[.NET Core Tool Installer](#)

[Node Tool Installer](#)

[Java Tool Installer](#)

[File matching patterns](#)

[File and variable transform](#)

[Permissions & security roles](#)

VSTS and Team Foundation Server help you implement a continuous integration (CI) and deployment (CD) pipeline for any app. Tutorials, references, and other documentation show you how to configure and manage CI/CD for the app and platform of your choice.

## 5-Minute quickstarts

### Learn how to build your app

[Android](#)  
[ASP.NET](#)  
[ASP.NET Core](#)  
[C/C++ with GCC](#)  
[C/C++ with VC++](#)  
[Docker image](#)



[Go](#)

[Gradle](#)  
 [Maven](#)  
 [.NET Desktop](#)  
 [Node.js](#)  
 [UWP](#)  
 [Xamarin](#)  
 [Xcode](#)

### Learn how to deploy your app

[Azure Web App](#)  
 [Linux VM](#)  
 [Web App for Containers](#)  
 [Windows VM](#)

### Learn how to test your app

[VS Test](#)

## Step-by-step tutorials

- [CI builds for Git in VSTS](#)
- [Set up multi-stage release](#)

## Videos

<a href="https://channel9.msdn.com/Events/Connect/2017/T174/player">https://channel9.msdn.com/Events/Connect/2017/T174/player</a>	<a href="https://channel9.msdn.com/Events/Connect/2017/T168/player">https://channel9.msdn.com/Events/Connect/2017/T168/player</a>
<a href="https://channel9.msdn.com/Events/Connect/2017/T170/player">https://channel9.msdn.com/Events/Connect/2017/T170/player</a>	<a href="https://channel9.msdn.com/Events/Connect/2017/T171/player">https://channel9.msdn.com/Events/Connect/2017/T171/player</a>
<a href="https://channel9.msdn.com/Events/Visual-Studio/Visual-Studio-2017-Launch/190/player">https://channel9.msdn.com/Events/Visual-Studio/Visual-Studio-2017-Launch/190/player</a>	

## Concepts

- [Release definitions](#)
- [Build and release agents](#)
- [Build and release tasks](#)
- [Licensing and build and release pipelines](#)

## Resources

- [What is continuous integration?](#)
- [What is continuous delivery?](#)
- [What is DevOps?](#)
- [Build and release marketplace extensions](#)

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

# Build and Release in VSTS and TFS

2/21/2018 • 2 min to read • [Edit Online](#)

Visual Studio Team Services (VSTS) is a collection of hosted DevOps services for application developers. Team Foundation Server (TFS) is the on-premises version of VSTS that you can install and manage on your own servers. Build and Release are two of the DevOps services in VSTS and TFS that help you manage continuous integration and delivery of your applications.

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 pipelines to drive frequent deployments. The Build service in VSTS and 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 pipelines 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 VSTS and TFS helps you set up and manage CD for your applications.

## 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. VSTS and TFS support two forms of version control - Git and Team Foundation Version Control. 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 Subversion, Bitbucket, GitHub, or any other Git repository. The Build service integrates with all of these version control systems.

## Application types

To configure CI, you create a build definition. A build 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. VSTS and TFS have 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

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. VSTS and TFS support deploying your application to virtual machines, containers, on-premises and cloud platforms, or PaaS services. You can also publish your mobile application to a store.

## Package formats

If your goal is to produce packages that can be consumed by others, VSTS has a built-in package management repository. You can package and publish your application bits as NuGet, NPM, or Maven packages into the built-in repository or into any other package management repository of your choice.

In the various topics covered in this documentation, you will explore how to configure CI for the version control repository and development language of your choice, configure CD to the deployment target of your choice, and how to publish packages in the packaging format of your choice.

# Deploy to Azure

2/21/2018 • 1 min to read • [Edit Online](#)

Visual Studio Team Services (VSTS) is Microsoft's DevOps solution for Azure. The Build and Release services provide streamlined experiences to deploy your apps to one of Azure's many services. These services provide a continuous delivery solution if your code is managed in GitHub, VSTS Git or Team Foundation Version Control, or in one of the other Git servers.

## Web apps

Azure web apps enable you to build and host web applications in the programming language of your choice without managing infrastructure. They offer auto-scaling and high availability and support both Windows and Linux. You can start directly from the Azure portal and set up an entire pipeline for your web app. You can then customize the build and release definitions in VSTS to meet the needs of your application.

[Build and Deploy to Web Apps](#)

## Virtual machines

Azure virtual machines provide on-demand, high-scale, secure, virtualized infrastructure using Windows, Red Hat, Ubuntu, or another Linux distribution of your choice. If you develop an application that runs on virtual machines, then VSTS has the tools to automate deployment of your app. You group the virtual machines in Azure into a deployment group in VSTS, and then set up a release definition to deploy to that group of virtual machines. You can also configure rolling deployment to ensure zero downtime for your customers.

[Deploy to Windows VMs](#)

## Web apps for containers

Azure web apps for containers offer the fastest and simplest way for you run a container instance in Azure. VSTS offers the simplest way for you to set up and track the continuous delivery of your container application. You configure your build process in VSTS to automatically publish a container image, and then your release process to automatically deploy your container image to an Azure web app. Furthermore, if you develop a ASP.NET Core application in Visual Studio, you can set up the entire VSTS build and release pipeline from Visual Studio IDE.

[CI/CD to Containers](#)

## And more

The Build and Release services in VSTS provide a number of tasks to provision your Azure infrastructure and to deploy your app to services such as Service Fabric or Kubernetes. Explore these possibilities using the various How-to topics in this documentation.

# Create a CI/CD pipeline for .NET with the Azure DevOps Project

2/21/2018 • 6 min to read • [Edit Online](#)

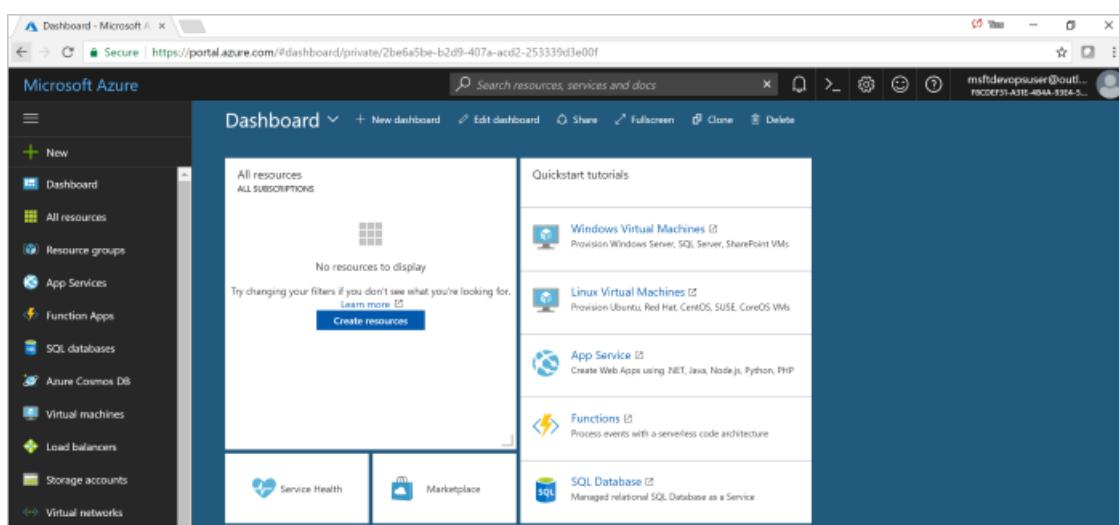
Configure continuous integration (CI) and continuous delivery (CD) for your .NET core or ASP.NET application with The **Azure DevOps Project**. The Azure DevOps project simplifies the initial configuration of a VSTS build and release pipeline.

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

## Sign in to the Azure portal

The Azure DevOps Project creates a CI/CD pipeline in VSTS. You can create a **new VSTS account** or use an **existing account**. The Azure DevOps Project also creates **Azure resources** in the **Azure subscription** of your choice.

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



## Select a sample application and Azure service

1. Select the **.NET** sample application. The .NET samples include a choice of either the open source ASP.NET framework or the cross-platform .NET Core framework.



Launch an app on any Azure Service of your choice in few quick steps.  
Everything you need, created and ready to go: code repository, CI/CD pipeline and all the necessary Azure resources.

New to DevOps Project?  
Check out our tutorials [↗](#)

Start fresh with a sample application

 .NET	 Java
New Web app by using ASP.NET or .NET Core	New Web app by using Spring or JSF
 Static Website	 Node.js
New static website by using HTML, CSS and JavaScript	New Web app by using Node.js, Express.js or Sails.js
 PHP	 Python
New Web app by using Laravel or Symphony	New Web app by using Python, Bottle, Django or Flask

or use your application

[Previous](#) [Next](#)

2. Select the **.NET Core** application framework. This sample is an ASP.NET Core MVC application. When you're done, choose **Next**.
3. **Web App on Windows** is the default deployment target. Optionally, you can choose Web App on Linux or Web App for Containers. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Select the **target service** of your choice. When you're done, choose **Next**.

## Configure VSTS and an Azure subscription

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



Runtime      Framework      Service      Create

## Almost there!

Ready to deploy an .NET Core web app to a new Web App on Windows.

### Visual Studio Team Services

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

\* Account

Create new  Use existing

devopstutorial



\* Name

SampleTeamProject



### Azure

[Change](#)

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

\* Subscription

Pay-As-You-Go



\* App Service Location

South Central US



\* Name

SampleASPNETCoreApp



.azurewebsites.net

Pricing tier: S1 Standard

[Previous](#)

[Done](#)

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

The screenshot shows the Azure DevOps project dashboard. On the left, there's a sidebar with 'CI/CD Pipeline' and 'Repository' sections. Under 'CI/CD Pipeline', there's a 'Code' section with a commit from 'SampleTeamProject' on 'master'. Below it is a 'Build' section for 'SampleASNETCoreApp' with a successful build. Under 'Production', there's another 'SampleASNETCoreApp' entry. In the main area, there's a 'Azure resources' section showing an 'Application endpoint' at <http://sampleaspnetcoreapp.azurewebsites.net>. An 'App Service' named 'SampleASNETCoreApp' is shown as 'Running'. Below that is an 'Application Insights' section with a timeline from 12 PM to 5 PM, showing a 'SERVER REQUEST' and a 'FAILED REQUEST'.

## Commit code changes and execute CI/CD

The Azure DevOps project created a Git repository in your VSTS or GitHub account. Follow the steps below to view the repository and make code changes to your application.

1. On the left-hand side of the DevOps project dashboard, select the link for your **master** branch. This link opens a view to the newly created Git repository.
2. To view the repository clone URL, select **Clone** from the top right of the browser. You can clone your Git repository in your favorite IDE. In the next few steps, you can use the web browser to make and commit code changes directly to the master branch.
3. On the left-hand side of the browser, navigate to the **Views/Home/index.cshtml** file.
4. Select **Edit**, and make a change to the h2 heading. For example, type **Get started right away with the Azure DevOps Project** or make some other change.

The screenshot shows a code editor with the file 'index.cshtml' open. The code contains an H2 tag with the text 'Get started right away with the Azure DevOps Project'. Above the code editor are tabs for 'Contents', 'History', 'Compare', and 'Blame', and a toolbar with 'Edit', 'Rename', and other icons.

```

1 <h2>
2   ViewBag.Title = "Home Page";
3 </h2>
4
5 <div class="main-container">
6   <div class="cloud-image">
7     
8   </div>
9   <div class="content">
10    <div class="tweet-container">
11      <a href="http://twitter.com/intent/tweet/?text=I%20just%20created%20a%20new%20ASP.net%20website%20on%20Azure%20using%20Azure%20DevOps%20Project">
12        
13      </a>
14    </div>
15    <div class="content-body">
16      <div class="success-text">Success!</div>
17      <div class="description line-1"> Azure DevOps Project has been successfully setup</div>
18      <div class="description line-2"> Your ASP.NET MVC app is up and running on Azure</div>
19    </div>
20  </div>
21 </div>
22
23 <div class="row">
24   <div class="col-md-4">
25     <h2>Get started right away with the Azure DevOps Project </h2>
26     <p>
27       | Clone your code repository and start developing your application on IDE of your choice
28     </p>
29     <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=862409">Learn more &gt;</a></p>
30   </div>

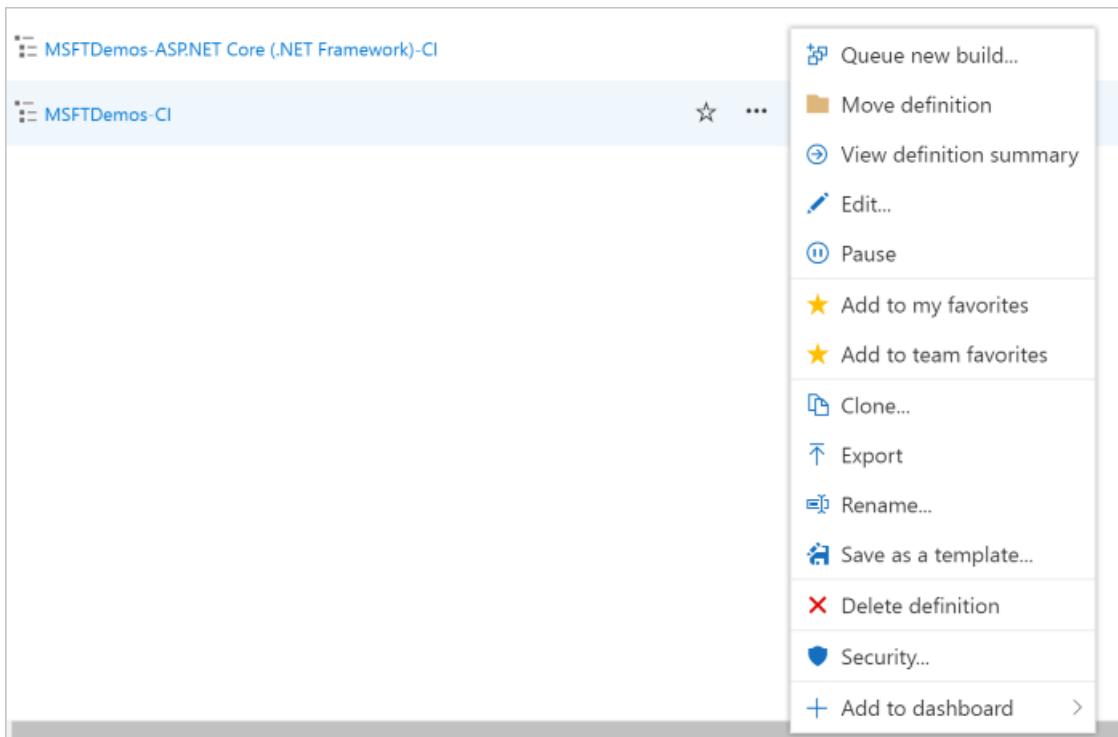
```

5. Choose **Commit**, then save your changes.
6. In your browser, navigate to the **Azure DevOps project dashboard**. You should now see a build is in progress. The changes you just made are automatically built and deployed via a VSTS CI/CD pipeline.

## Examine the VSTS CI/CD pipeline

The Azure DevOps project automatically configured a full VSTS CI/CD pipeline in your VSTS account. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release definitions.

1. Select **Build Pipelines** from the **top** of the Azure DevOps project dashboard. This link opens a browser tab and opens the VSTS build definition for your new project.
2. Select the **ellipsis**. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build definition.
3. Select **Edit**.



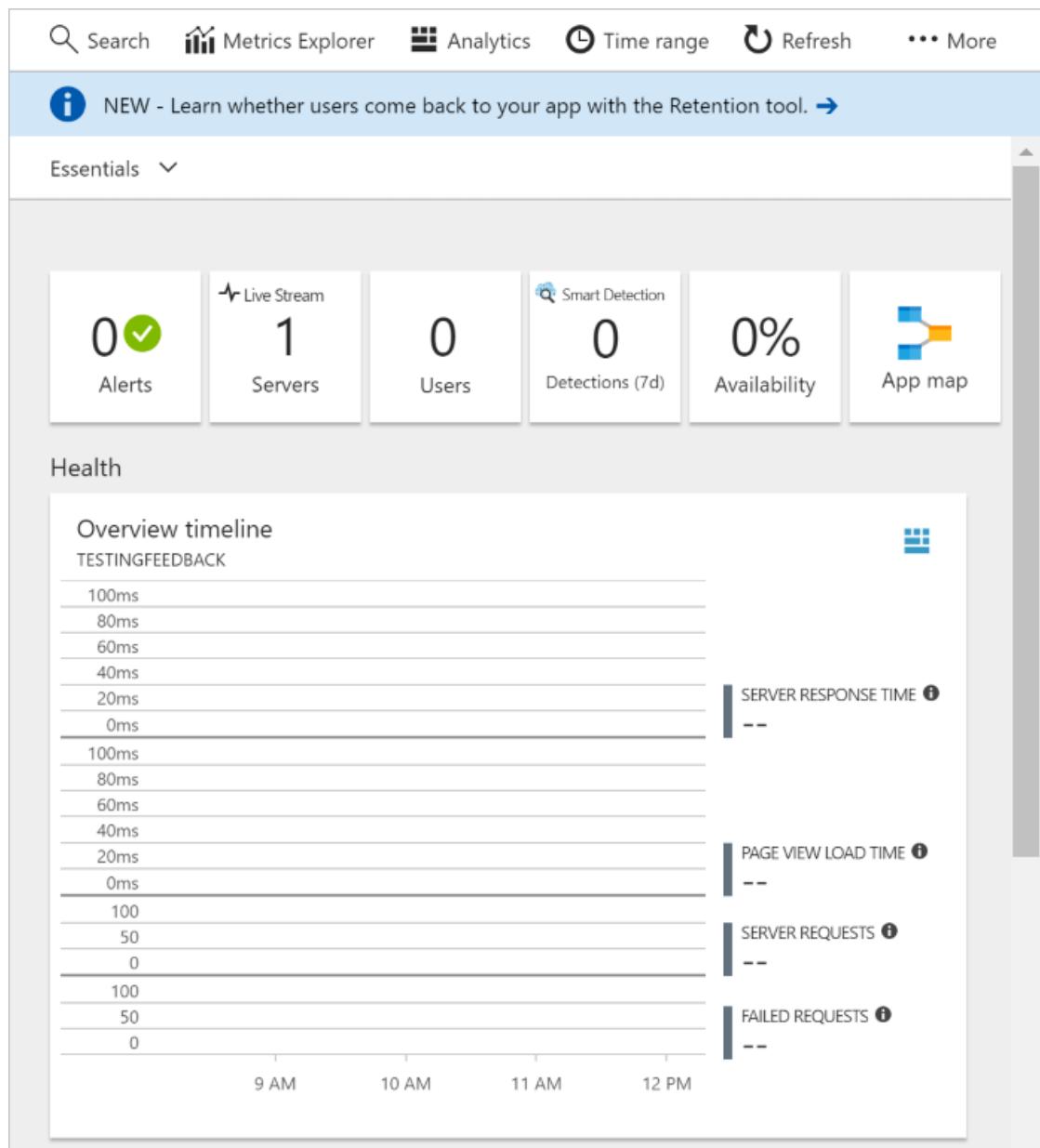
4. From this view, **examine the various tasks** for your build definition. The build performs various tasks such as fetching sources from the Git repository, restoring dependencies, and publishing outputs used for deployments.
5. At the top of the build definition, select the **build definition name**.
6. Change the **name** of your build definition to something more descriptive. Select **Save & queue**, then select **Save**.
7. Under your build definition name, select **History**. You see an audit trail of your recent changes for the build. VSTS keeps track of any changes made to the build definition, and allows you to compare versions.
8. Select **Triggers**. The Azure DevOps project automatically created a CI trigger, and every commit to the repository initiates a new build. You can optionally choose to include or exclude branches from the CI process.
9. Select **Retention**. Based on your scenario, you can specify policies to keep or remove a certain number of builds.
10. Select **Build and Release**, then choose **Releases**. The Azure DevOps project created a VSTS release definition to manage deployments to Azure.
11. On the left-hand side of the browser, select the **ellipsis** next to your release definition, then choose **Edit**.

12. The release definition contains a **Pipeline**, which defines the release process. Under **Artifacts**, select **Drop**. The build definition you examined in the previous steps produces the output used for the artifact.
13. To the right-hand side of the **Drop** icon, select the **Continuous deployment trigger**. This release definition has an enabled CD trigger, which executes a deployment every time there is a new build artifact available. Optionally, you can disable the trigger, so your deployments require manual execution.
14. On the left-hand side of the browser, select **Tasks**. The tasks are the activities your deployment process performs. In this example, a task was created to deploy to **Azure App service**.
15. On the right-hand side of the browser, select **View releases**. This view shows a history of releases.
16. Select the **ellipsis** next to one of your releases, and choose **Open**. There are several menus to explore from this view such as a release summary, associated work items, and tests.
17. Select **Commits**. This view shows code commits associated with the specific deployment.
18. Select **Logs**. The logs contain useful information about the deployment process. They can be viewed both during and after deployments.

## Configure Azure Application Insights monitoring

With Azure Application insights, you can easily monitor your application's performance and usage. The Azure DevOps project automatically configured an Application Insights resource for your application. You can further configure various alerts and monitoring capabilities as needed.

1. Navigate to the **Azure DevOps Project** dashboard in the Azure portal. On the bottom-right of the dashboard, choose the **Application Insights** link for your app.
2. The **Application Insights** blade opens in the Azure portal. This view contains usage, performance, and availability monitoring information for your app.



3. Select **Time range**, and then choose **Last hour**. Select **Update** to filter the results. You now see all activity from the last 60 minutes. Select the **x** to exit time range.
4. Select **Alerts**, then select **+ Add metric alert**.
5. Enter a **Name** for the alert.
6. Select the drop-down for **Source Alter on**. Choose your **App Service resource**.
7. The default alert is for a **server response time greater than 1 second**. Select the **Metric** drop-down to examine the various alert metrics. You can easily configure a variety of alerts to improve the monitoring capabilities of your app.
8. Select the check-box for **Notify via Email owners, contributors, and readers**. Optionally, you can perform additional actions when an alert fires by executing an Azure logic app.
9. Choose **Ok** to create the alert. In a few moments, the alert appears as active on the dashboard. **Exit** the Alerts area, and navigate back to the **Application Insights blade**.
10. Select **Availability**, then select **+ Add test**.
11. Enter a **Test name**, then choose **Create**. A simple ping test is created to verify the availability of your application. After a few minutes, test results are available, and the Application Insights dashboard displays an availability status.

## Clean up resources

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

## Next steps

To learn more about modifying the build and release definitions to meet the needs of your team, see this tutorial:

[Customize CD process](#)

# Create a CI/CD pipeline for Node.js with the Azure DevOps Project

2/21/2018 • 7 min to read • [Edit Online](#)

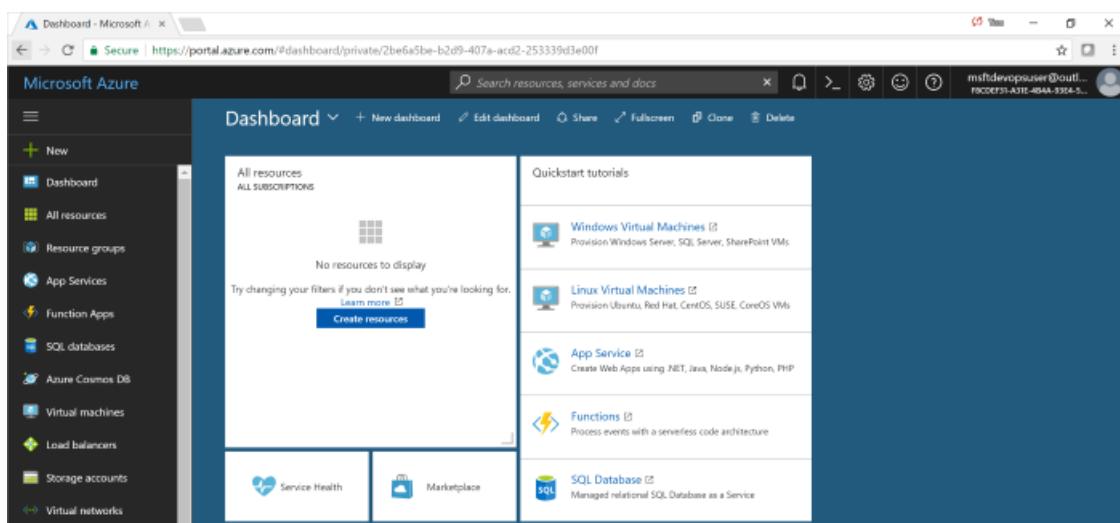
The Azure DevOps Project presents a simplified experience which creates Azure resources and sets up a continuous integration (CI) and continuous delivery (CD) pipeline for your Node.js app in Visual Studio Team Services (VSTS), which is Microsoft's DevOps solution for Azure.

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

## Sign in to the Azure portal

The Azure DevOps Project creates a CI/CD pipeline in VSTS. You can create a free **new VSTS** account or use an **existing account**. The DevOps Project also creates **Azure resources** in the **Azure subscription** of your choice.

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



## Select a sample application and Azure service

1. Select the **Node.js** sample application. The Node.js samples include a choice of several application frameworks.
2. The default sample framework is **Express.js**. Select a framework or leave the default setting. When you're done, choose **Next**.
3. **Web App on Windows** is the default deployment target. Optionally, you can choose Web App on Linux or Web App for Containers. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Select the **target service** of your choice. When you're done, choose **Next**.

## Configure VSTS and an Azure subscription

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



## Almost there!

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

### Visual Studio Team Services

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

\* Account

Create new  Use existing

devopstutorial

\* Name

nodesample

### Azure

[Change](#)

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

\* Subscription

Pay-As-You-Go

\* App Service Location

South Central US

\* Name

nodesamplesite



.azurewebsites.net

Pricing tier: S1 Standard

[Previous](#)

[Done](#)

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

The screenshot shows the Azure DevOps project dashboard. On the left, under 'CI/CD Pipeline', there's a flowchart from 'Code' to 'Build' to 'Production'. 'Code' shows a commit by 'Alok Agrawal' (3d ago). 'Build' shows a build step for 'nodesampleprojectsite' (Build 20171109.1, Succeeded, 1 min ago). 'Production' shows a deployment step for 'nodesampleprojectsite' (Production, In Progress, 1 min ago). Under 'Repository', it shows a 'nodesampleproject' with 1 commit in the past 7 days. On the right, under 'Azure resources', it shows an 'Application endpoint' at <http://nodesampleprojectsites.azurewebsites.net>. An 'App Service' named 'nodesampleprojectsites' is listed as 'Running'. Under 'Application Insights', there's a timeline from 3 PM to 8 PM showing a 'SERVER REQUEST' and a 'FAILED REQUEST'. A 'Code' button is also present.

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

## Commit code changes and execute CI/CD

The Azure DevOps project created a Git repository in your VSTS or GitHub account. Follow the steps below to view the repository and make code changes to your application.

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

## Examine the VSTS CI/CD pipeline

The Azure DevOps project automatically configured a full VSTS CI/CD pipeline in your VSTS account. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release definitions.

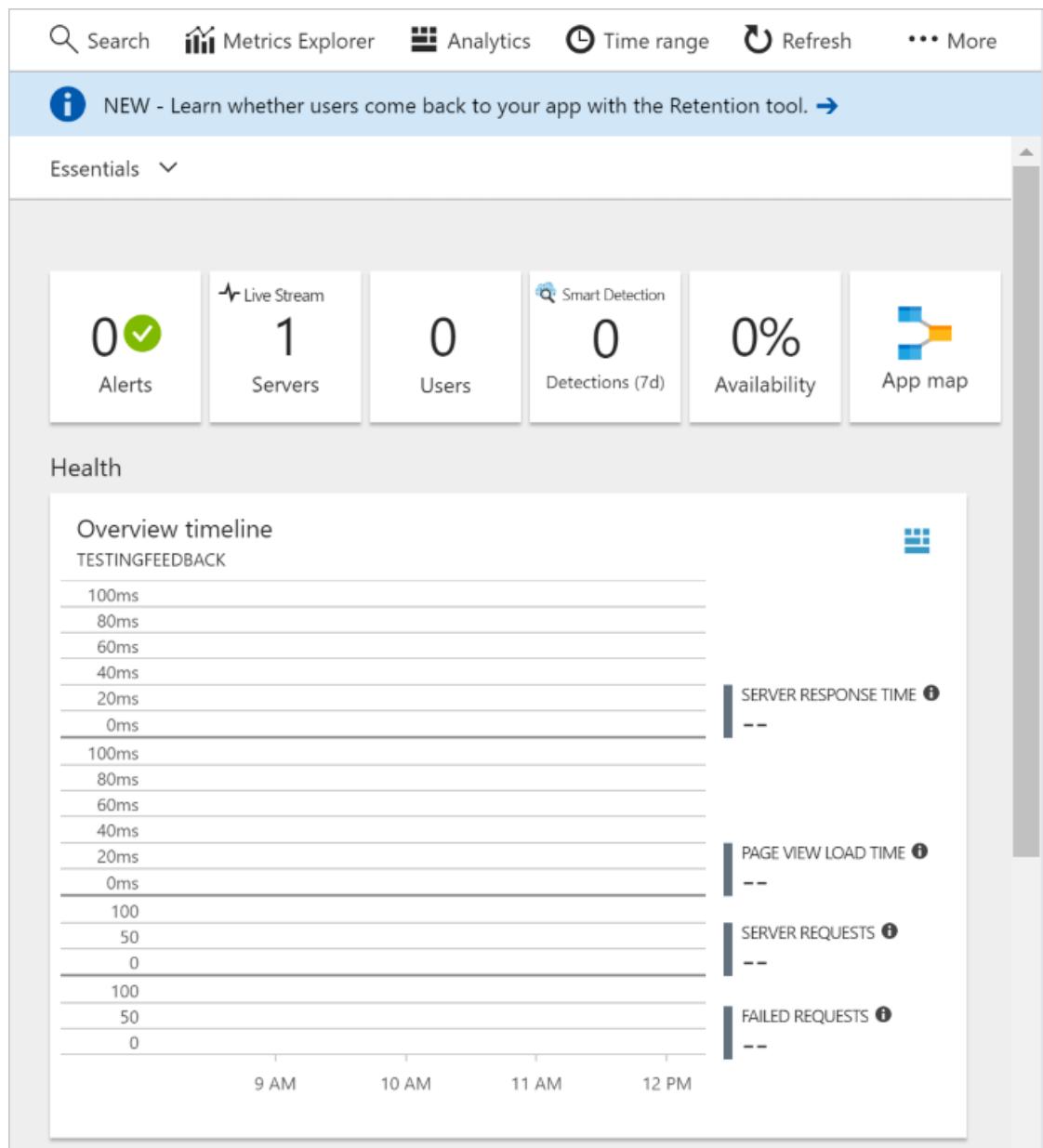
1. Select **Build Pipelines** from the **top** of the Azure DevOps project dashboard. This link opens a browser tab and opens the VSTS build definition for your new project.
2. Move the mouse cursor to the right of the build definition next to the **Status** field. Select the **ellipsis** that appears. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build definition.

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

## Configure Azure Application Insights monitoring

With Azure Application insights, you can easily monitor your application's performance and usage. The Azure DevOps project automatically configured an Application Insights resource for your application. You can further configure various alerts and monitoring capabilities as needed.

1. Navigate to the **Azure DevOps Project** dashboard in the Azure portal. On the bottom-right of the dashboard, choose the **Application Insights** link for your app.
2. The **Application Insights** blade opens in the Azure portal. This view contains usage, performance, and availability monitoring information for your app.



3. Select **Time range**, and then choose **Last hour**. Select **Update** to filter the results. You now see all activity from the last 60 minutes. Select the **x** to exit time range.
4. Select **Alerts**, then select **+ Add metric alert**.
5. Enter a **Name** for the alert.
6. Select the drop-down for **Source Alter on**. Choose your **App Service resource**.
7. The default alert is for a **server response time greater than 1 second**. Select the **Metric** drop-down to examine the various alert metrics. You can easily configure a variety of alerts to improve the monitoring capabilities of your app.
8. Select the check-box for **Notify via Email owners, contributors, and readers**. Optionally, you can perform additional actions when an alert fires by executing an Azure logic app.
9. Choose **Ok** to create the alert. In a few moments, the alert appears as active on the dashboard. **Exit** the Alerts area, and navigate back to the **Application Insights blade**.
10. Select **Availability**, then select **+ Add test**.
11. Enter a **Test name**, then choose **Create**. A simple ping test is created to verify the availability of your application. After a few minutes, test results are available, and the Application Insights dashboard displays an availability status.

## Clean up resources

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

## Next steps

When you configured your CI/CD process in this quickstart, a build and release definition were automatically created in your VSTS project. You can modify these build and release definitions to meet the needs of your team. To learn more see this tutorial:

[Customize CD process](#)

# Create a CI/CD pipeline for Java with the Azure DevOps Project

2/21/2018 • 6 min to read • [Edit Online](#)

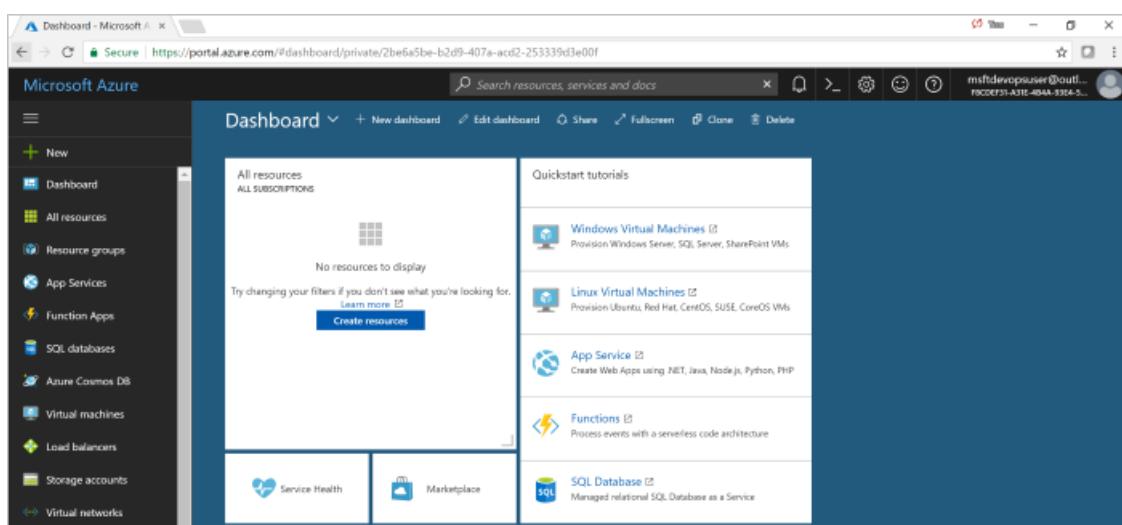
The Azure DevOps Project presents a simplified experience which creates Azure resources and sets up a continuous integration (CI) and continuous delivery (CD) pipeline for your Java app in Visual Studio Team Services (VSTS), which is Microsoft's DevOps solution for Azure.

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

## Sign in to the Azure portal

The Azure DevOps Project creates a CI/CD pipeline in VSTS. You can create a free **new VSTS** account or use an **existing account**. The DevOps Project also creates **Azure resources** in the **Azure subscription** of your choice.

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



## Select a sample application and Azure service

1. Select the **Java** sample application. The Java samples include a choice of several application frameworks.
2. The default sample framework is **Spring**. Select a framework or leave the default setting. When you're done, choose **Next**.
3. **Web App For Containers** is the default deployment target. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Select the **target service** of your choice. When you're done, choose **Next**.

## Configure VSTS and an Azure subscription

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

## DEPLOY TO:

Ready to deploy Spring web app to a new Web App for Containers.

### Visual Studio Team Services

Change

Visual Studio Team Services (VSTS) for building and deploying your app

\* Account

Create new  Use existing

devopsprojecttester



.visualstudio.com

\* Project name

sample



### Azure

Change

Azure resources required for running your app

\* Subscription

Pay-As-You-Go



\* App name

sampleappsite



.azurewebsites.net

\* App Service location

South Central US



Pricing tier: S1 Standard

By continuing, you agree to the [Terms of Service](#) and the [Privacy Statement](#). The new app code is published under the MIT license.

[Previous](#)

[Done](#)

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

The screenshot shows the Azure DevOps project dashboard. On the left, the 'CI/CD Pipeline' section displays a flow from 'Code' (sample branch, master) through 'Build' (sampleappsite - CI) to 'Dev' (sampleappsite - CD). The 'Build' step shows 'Build 20171221.1' in progress. On the right, the 'Azure resources' section shows an 'Application endpoint' at <http://sampleappsite.azurewebsites.net> with a 'Browse' button. The 'App Service' section shows 'sampleappsite' is 'Running'. Below that is the 'Application Insights' section, which includes a chart showing 'SERVER REQUEST' and 'FAILED REQUEST' over time from 3 PM to 8 PM.

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

## Commit code changes and execute CI/CD

The Azure DevOps project created a Git repository in your VSTS or GitHub account. Follow the steps below to view the repository and make code changes to your application.

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

## Examine the VSTS CI/CD pipeline

The Azure DevOps project automatically configured a full VSTS CI/CD pipeline in your VSTS account. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release definitions.

1. Select **Build Pipelines** from the **top** of the Azure DevOps project dashboard. This link opens a browser tab and opens the VSTS build definition for your new project.

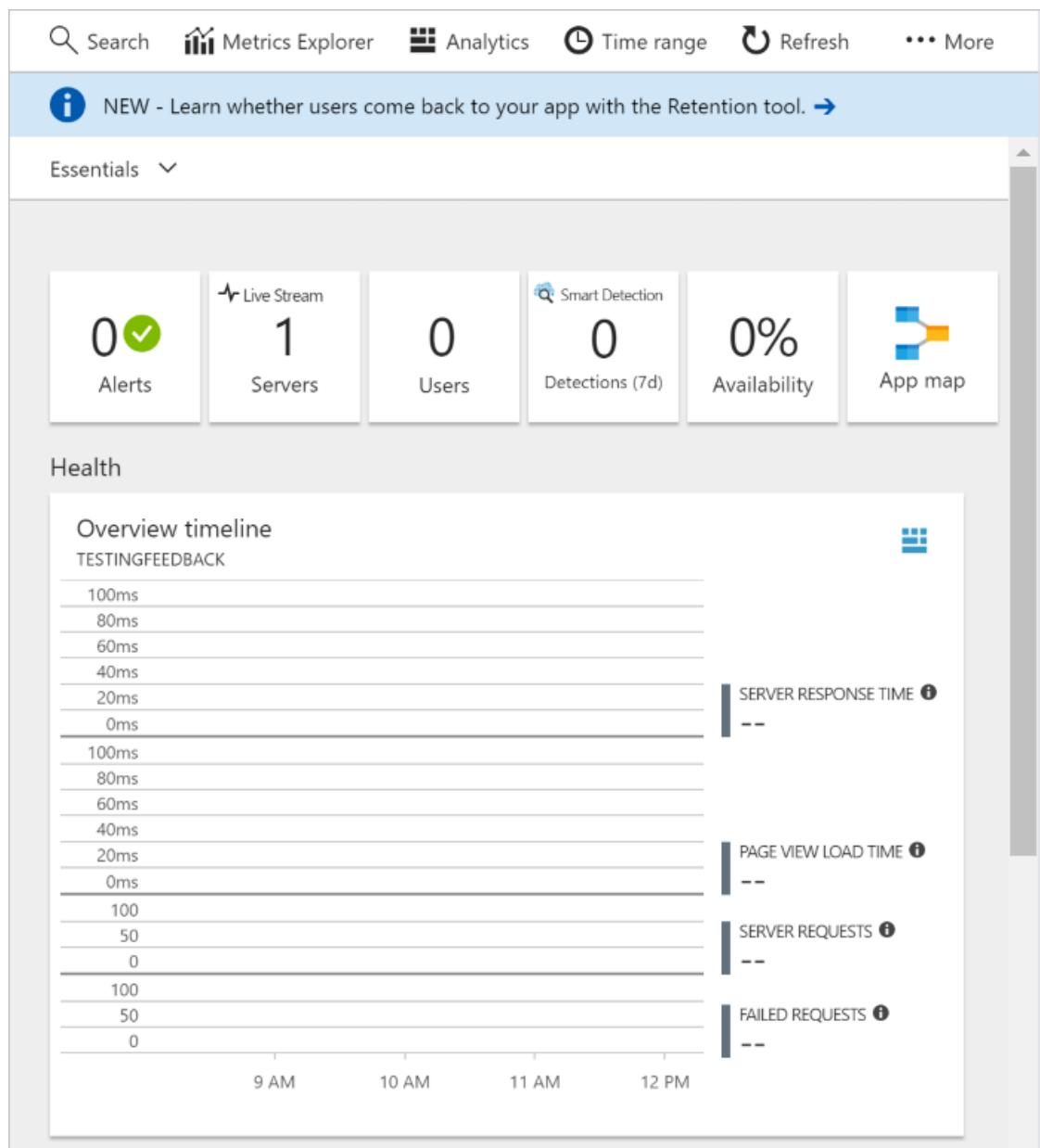
2. Move the mouse cursor to the right of the build definition next to the **Status** field. Select the **ellipsis** that appears. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build definition.
3. Select **Edit**.
4. From this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

## Configure Azure Application Insights monitoring

With Azure Application insights, you can easily monitor your application's performance and usage. The Azure DevOps project automatically configured an Application Insights resource for your application. You can further configure various alerts and monitoring capabilities as needed.

1. Navigate to the **Azure DevOps Project** dashboard in the Azure portal. On the bottom-right of the dashboard, choose the **Application Insights** link for your app.
2. The **Application Insights** blade opens in the Azure portal. This view contains usage, performance, and

availability monitoring information for your app.



3. Select **Time range**, and then choose **Last hour**. Select **Update** to filter the results. You now see all activity from the last 60 minutes. Select the **x** to exit time range.
4. Select **Alerts**, then select **+ Add metric alert**.
5. Enter a **Name** for the alert.
6. Select the drop-down for **Source Alter on**. Choose your **App Service resource**.
7. The default alert is for a **server response time greater than 1 second**. Select the **Metric** drop-down to examine the various alert metrics. You can easily configure a variety of alerts to improve the monitoring capabilities of your app.
8. Select the check-box for **Notify via Email owners, contributors, and readers**. Optionally, you can perform additional actions when an alert fires by executing an Azure logic app.
9. Choose **Ok** to create the alert. In a few moments, the alert appears as active on the dashboard. **Exit** the Alerts area, and navigate back to the **Application Insights blade**.
10. Select **Availability**, then select **+ Add test**.
11. Enter a **Test name**, then choose **Create**. A simple ping test is created to verify the availability of your

application. After a few minutes, test results are available, and the Application Insights dashboard displays an availability status.

## Clean up resources

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

## Next steps

When you configured your CI/CD process in this quickstart, a build and release definition were automatically created in your VSTS project. You can modify these build and release definitions to meet the needs of your team. To learn more see this tutorial:

[Customize CD process](#)

# Create a CI/CD pipeline for Python with the Azure DevOps Project

2/21/2018 • 6 min to read • [Edit Online](#)

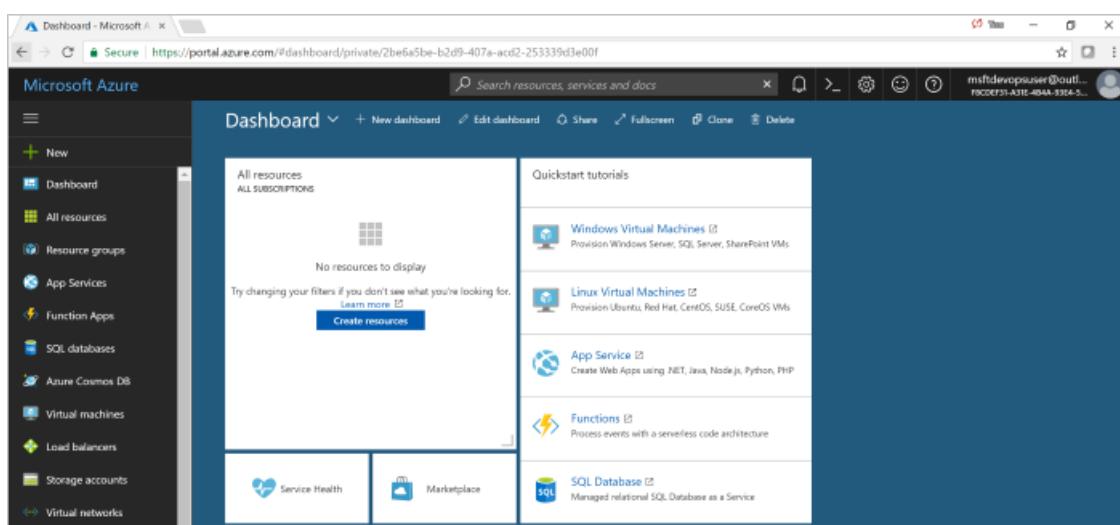
The Azure DevOps Project presents a simplified experience which creates Azure resources and sets up a continuous integration (CI) and continuous delivery (CD) pipeline for your Python app in Visual Studio Team Services (VSTS), which is Microsoft's DevOps solution for Azure.

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

## Sign in to the Azure portal

The Azure DevOps Project creates a CI/CD pipeline in VSTS. You can create a free **new VSTS** account or use an **existing account**. The DevOps Project also creates **Azure resources** in the **Azure subscription** of your choice.

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



## Select a sample application and Azure service

1. Select the **Python** sample application. The Python samples include a choice of several application frameworks.
2. The default sample framework is **Django**. Select a framework or leave the default setting. When you're done, choose **Next**.
3. **Web App For Containers** is the default deployment target. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Select the **target service** of your choice. When you're done, choose **Next**.

## Configure VSTS and an Azure subscription

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

## Almost there!

Ready to deploy Spring web app to a new Web App for Containers.

### Visual Studio Team Services

[Change](#)

Visual Studio Team Services (VSTS) for building and deploying your app

\* Account

Create new  Use existing

devopsprojecttester



.visualstudio.com

\* Project name

sample



### Azure

[Change](#)

Azure resources required for running your app

\* Subscription

Pay-As-You-Go



\* App name

sampleappsite



.azurewebsites.net

\* App Service location

South Central US



Pricing tier: S1 Standard

By continuing, you agree to the [Terms of Service](#) and the [Privacy Statement](#). The new app code is published under the MIT license.

[Previous](#)

[Done](#)

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

The screenshot shows the Azure DevOps project dashboard. On the left, the 'CI/CD Pipeline' section displays a flow from 'Code' (sample master branch) through 'Build' (sampleappsite - CI) to 'Dev' (sampleappsite - CD). The 'Build' step shows a 'Build 20171221.1' task in progress. On the right, the 'Azure resources' section shows the 'Application endpoint' as <http://sampleappsite.azurewebsites.net>. Below it, the 'App Service' section shows 'sampleappsite' is 'Running'. Under 'Application Insights', there is a chart with two data series: 'SERVER REQUEST' and 'FAILED REQUEST', both of which show a single data point at 3 PM.

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

## Commit code changes and execute CI/CD

The Azure DevOps project created a Git repository in your VSTS or GitHub account. Follow the steps below to view the repository and make code changes to your application.

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

## Examine the VSTS CI/CD pipeline

The Azure DevOps project automatically configured a full VSTS CI/CD pipeline in your VSTS account. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release definitions.

1. Select **Build Pipelines** from the **top** of the Azure DevOps project dashboard. This link opens a browser tab

and opens the VSTS build definition for your new project.

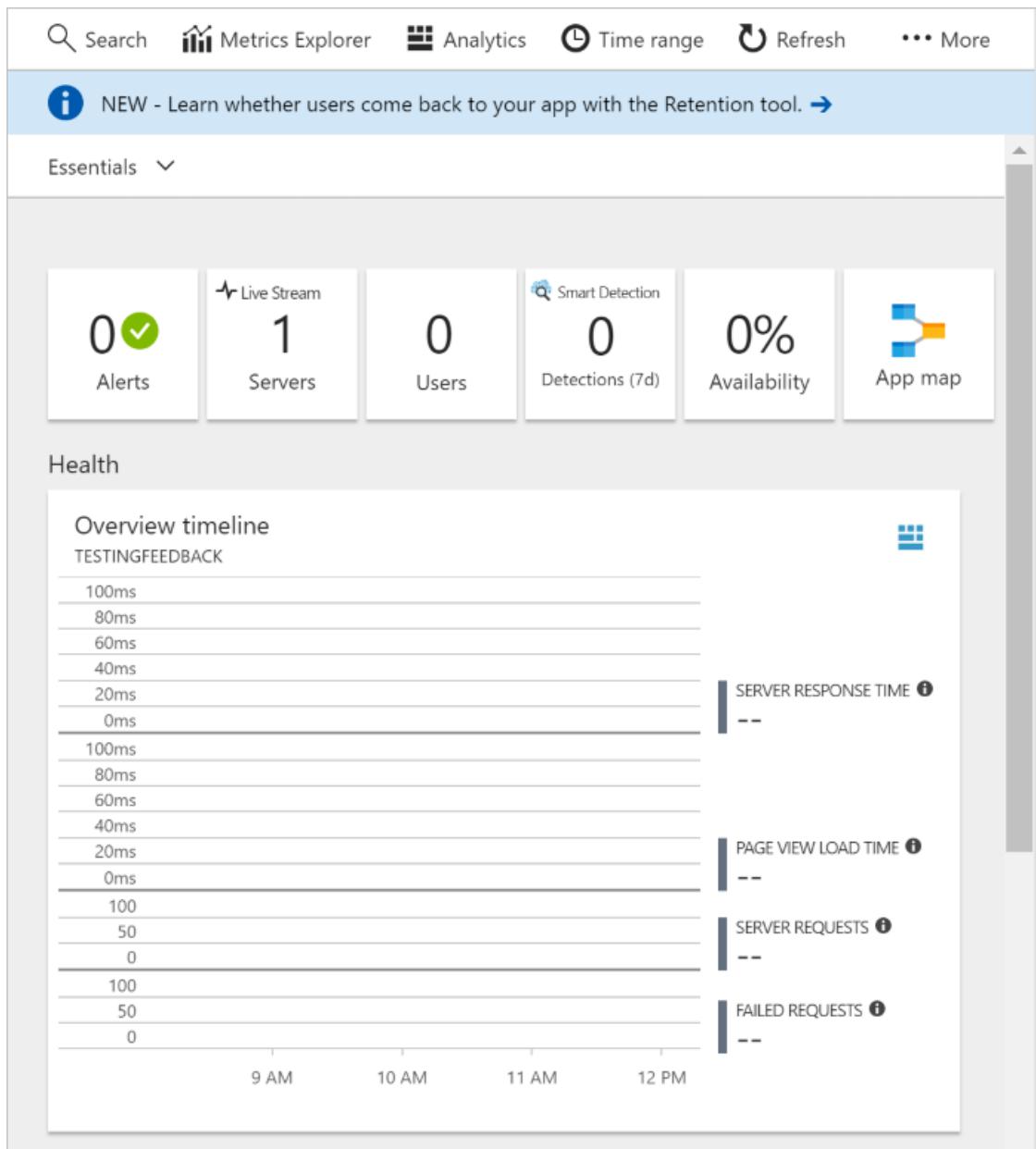
2. Move the mouse cursor to the right of the build definition next to the **Status** field. Select the **ellipsis** that appears. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build definition.
3. Select **Edit**.
4. From this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

## Configure Azure Application Insights monitoring

With Azure Application insights, you can easily monitor your application's performance and usage. The Azure DevOps project automatically configured an Application Insights resource for your application. You can further configure various alerts and monitoring capabilities as needed.

1. Navigate to the **Azure DevOps Project** dashboard in the Azure portal. On the bottom-right of the dashboard, choose the **Application Insights** link for your app.

2. The **Application Insights** blade opens in the Azure portal. This view contains usage, performance, and availability monitoring information for your app.



3. Select **Time range**, and then choose **Last hour**. Select **Update** to filter the results. You now see all activity from the last 60 minutes. Select the **x** to exit time range.
4. Select **Alerts**, then select **+ Add metric alert**.
5. Enter a **Name** for the alert.
6. Select the drop-down for **Source Alter on**. Choose your **App Service resource**.
7. The default alert is for a **server response time greater than 1 second**. Select the **Metric** drop-down to examine the various alert metrics. You can easily configure a variety of alerts to improve the monitoring capabilities of your app.
8. Select the check-box for **Notify via Email owners, contributors, and readers**. Optionally, you can perform additional actions when an alert fires by executing an Azure logic app.
9. Choose **Ok** to create the alert. In a few moments, the alert appears as active on the dashboard. **Exit** the Alerts area, and navigate back to the **Application Insights blade**.
10. Select **Availability**, then select **+ Add test**.

11. Enter a **Test name**, then choose **Create**. A simple ping test is created to verify the availability of your application. After a few minutes, test results are available, and the Application Insights dashboard displays an availability status.

## Clean up resources

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

## Next steps

When you configured your CI/CD process in this quickstart, a build and release definition were automatically created in your VSTS project. You can modify these build and release definitions to meet the needs of your team. To learn more see this tutorial:

[Customize CD process](#)

# Create a CI/CD pipeline for PHP with the Azure DevOps Project

2/21/2018 • 6 min to read • [Edit Online](#)

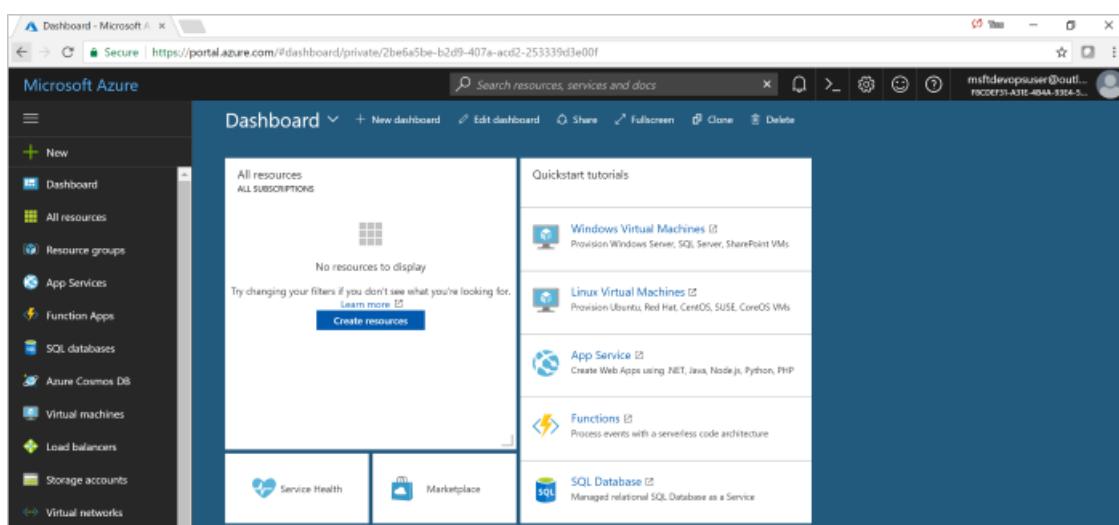
The Azure DevOps Project presents a simplified experience which creates Azure resources and sets up a continuous integration (CI) and continuous delivery (CD) pipeline for your PHP app in Visual Studio Team Services (VSTS), which is Microsoft's DevOps solution for Azure.

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

## Sign in to the Azure portal

The Azure DevOps Project creates a CI/CD pipeline in VSTS. You can create a free **new VSTS** account or use an **existing account**. The DevOps Project also creates **Azure resources** in the **Azure subscription** of your choice.

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



## Select a sample application and Azure service

1. Select the **PHP** sample application. The PHP samples include a choice of several application frameworks.
2. The default sample framework is **Laravel**. Select a framework or leave the default setting. When you're done, choose **Next**.
3. **Web App For Containers** is the default deployment target. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Select the **target service** of your choice. When you're done, choose **Next**.

## Configure VSTS and an Azure subscription

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

## DEPLOY TO:

Ready to deploy Spring web app to a new Web App for Containers.

### Visual Studio Team Services

Change

Visual Studio Team Services (VSTS) for building and deploying your app

\* Account

Create new  Use existing

devopsprojecttester



.visualstudio.com

\* Project name

sample



### Azure

Change

Azure resources required for running your app

\* Subscription

Pay-As-You-Go



\* App name

sampleappsite



.azurewebsites.net

\* App Service location

South Central US



Pricing tier: S1 Standard

By continuing, you agree to the [Terms of Service](#) and the [Privacy Statement](#). The new app code is published under the MIT license.

[Previous](#)

[Done](#)

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

The screenshot shows the Azure DevOps project dashboard. On the left, the 'CI/CD Pipeline' section displays a flow from 'Code' (sample branch, master) through 'Build' (sampleappsite - CI) to 'Dev' (sampleappsite - CD). The 'Build' step shows 'Build 20171221.1' in progress. On the right, the 'Azure resources' section shows an 'Application endpoint' at <http://sampleappsite.azurewebsites.net> (Browse button), an 'App Service' named 'sampleappsite' (Running status), and 'Application Insights' showing a timeline from 3 PM to 8 PM with one 'SERVER REQUEST' and one 'FAILED REQUEST'. Below the pipeline is the 'Repository' section, which shows the 'sample' repository with no commits in the past 7 days and a 'Code' button.

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

## Commit code changes and execute CI/CD

The Azure DevOps project created a Git repository in your VSTS or GitHub account. Follow the steps below to view the repository and make code changes to your application.

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

## Examine the VSTS CI/CD pipeline

The Azure DevOps project automatically configured a full VSTS CI/CD pipeline in your VSTS account. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release definitions.

1. Select **Build Pipelines** from the **top** of the Azure DevOps project dashboard. This link opens a browser tab and opens the VSTS build definition for your new project.

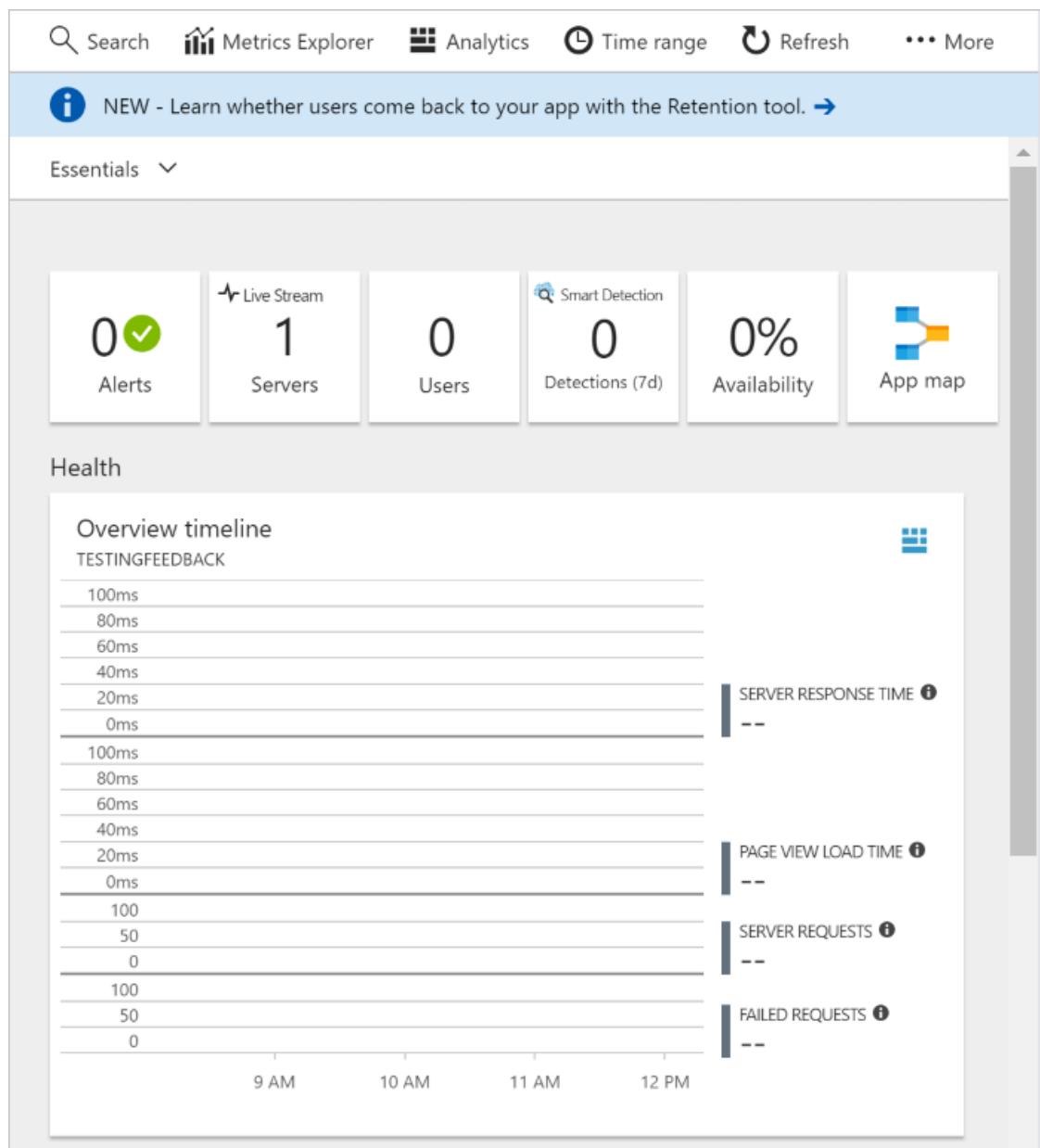
2. Move the mouse cursor to the right of the build definition next to the **Status** field. Select the **ellipsis** that appears. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build definition.
3. Select **Edit**.
4. From this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

## Configure Azure Application Insights monitoring

With Azure Application insights, you can easily monitor your application's performance and usage. The Azure DevOps project automatically configured an Application Insights resource for your application. You can further configure various alerts and monitoring capabilities as needed.

1. Navigate to the **Azure DevOps Project** dashboard in the Azure portal. On the bottom-right of the dashboard, choose the **Application Insights** link for your app.
2. The **Application Insights** blade opens in the Azure portal. This view contains usage, performance, and

availability monitoring information for your app.



3. Select **Time range**, and then choose **Last hour**. Select **Update** to filter the results. You now see all activity from the last 60 minutes. Select the **x** to exit time range.
4. Select **Alerts**, then select **+ Add metric alert**.
5. Enter a **Name** for the alert.
6. Select the drop-down for **Source Alter on**. Choose your **App Service resource**.
7. The default alert is for a **server response time greater than 1 second**. Select the **Metric** drop-down to examine the various alert metrics. You can easily configure a variety of alerts to improve the monitoring capabilities of your app.
8. Select the check-box for **Notify via Email owners, contributors, and readers**. Optionally, you can perform additional actions when an alert fires by executing an Azure logic app.
9. Choose **Ok** to create the alert. In a few moments, the alert appears as active on the dashboard. **Exit** the Alerts area, and navigate back to the **Application Insights blade**.
10. Select **Availability**, then select **+ Add test**.
11. Enter a **Test name**, then choose **Create**. A simple ping test is created to verify the availability of your

application. After a few minutes, test results are available, and the Application Insights dashboard displays an availability status.

## Clean up resources

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

## Next steps

When you configured your CI/CD process in this quickstart, a build and release definition were automatically created in your VSTS project. You can modify these build and release definitions to meet the needs of your team. To learn more see this tutorial:

[Customize CD process](#)

# Build your Android app

1/26/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build and package your Android app whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead.

First, you will need a build agent on which the Android SDK is installed. You may use one of the following:

1. The **Hosted VS2017** agent provided by VSTS, or
2. See [Build and release agents](#) for instructions on creating your own Linux, macOS, or Windows build agent. The Android SDK must be installed on your agent.

## Get the Android sample code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

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

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the 'MyFirstProject' dashboard in the Azure DevOps interface. The top navigation bar includes 'MyFirstProject', 'Dashboards', 'Code', '...', 'Search work items', and user profile icons. Below the navigation is a header for 'SampleApp' with tabs for 'Files' (selected), 'History', 'Branches', 'Tags', and 'Pull Requests'. A 'Clone' button is also present. The main content area shows a file tree for 'master' under 'SampleApp' with a search bar. A prominent blue button labeled 'Set up build' is highlighted with a red border.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, click **Android**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**, select **Hosted VS2017** or a queue that includes the agent you set up.

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
6. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
7. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows the build summary for 'Build 20170828.2'. The left sidebar lists the build steps: 'Build 20170828.2' (selected), 'Build', and 'Initialize Agent', each with a green checkmark. The main pane shows the build details: 'MyFirstProject-Cl / Build 20170828.2 / Build' (highlighted with a red box). Below this are links to 'Edit build definition' and 'Queue new build...'. A large green banner at the bottom states 'Build succeeded'.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

1. To sign your Android app using a keystore file as part of CI, see [Sign your mobile app](#).
2. Your Android app can be tested on physical devices using the **App Center Test** task.
3. You can distribute your Android app using the **App Center Distribute** task or tasks in the [Google Play extension](#).

# Build your ASP.NET Core app

2/21/2018 • 11 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.3

Follow these steps to set up a continuous integration (CI) process for an ASP.NET Core app using Visual Studio Team Services (VSTS) or Team Foundation Server (TFS).

As you walk through this quickstart, we'll ask you to choose:

- Which kind of Git service you're using: VSTS or TFS, or GitHub.
- How you want to define your build: in a web interface, or configured as code in YAML
- For continuous deployment, what is your target: Azure web app or IIS server in a Windows VM, a Linux VM, or a Docker container.

As you choose from these options in the sections below, this topic will adapt to your choices.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use TFS. Make sure that you have [configured a build agent](#) for your team project, and that you have Visual Studio 2017 installed on the agent machine and that you installed the latest Visual Studio update.

## Get the sample code

The sample app we use here is a Visual Studio solution that has two projects: An ASP.NET Core Web Application project and a Unit Test project (both targeting .NET Core 2.0 framework). This quickstart works for any apps that target the .NET Core 1.1 or 2.0 frameworks.

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

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

Where do you want to keep your code? Whichever service you choose, our system can automatically clone and pull code from it every time you push a change.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

You can also use Team Foundation Version Control (TFVC), Subversion, Bitbucket, or any other Git

repository for managing your source code, and still use VSTS for CI.

## Web or config as code

Do you want to define your build process in your web browser or configure it as code in YAML?

- [Web](#)
- [YAML](#)

Choose this option if you prefer a graphical interface in your web browser.



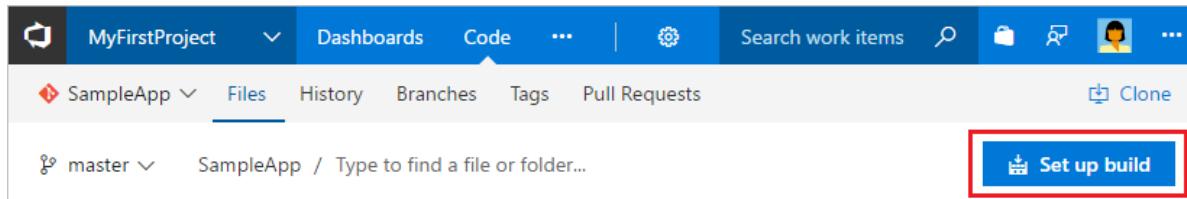
## Create the CI process definition

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

Begin by creating your build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

1. Navigate to the **Files** tab of the **Code** hub, and then choose **Set up build**.



You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, select **ASP.NET Core**, and then choose **Apply**.

The screenshot shows the 'Build & Release' tab selected in the top navigation bar. Below it, a search bar and several navigation links like 'Builds', 'Releases', 'Packages', 'Library', 'Task Groups', and 'Deployment Groups\*' are visible. The main area is titled 'Select a template' with a large 'Featured' section. It lists three templates: '.NET Desktop', 'ASP.NET (PREVIEW)', and 'ASP.NET Core'. The 'ASP.NET Core' section is highlighted with a red border around its 'Apply' button. A large circular arrow icon is on the left, and a note below it says: 'Choose a template that builds your kind of app. Don't worry if it's not an exact match; you can add and customize the tasks later.'

## Choose your deployment target

While a CI build process is a powerful way to do day-to-day development, continuous deployment is how many teams accelerate how they deliver value to customers. After each successful CI build, you can automatically deploy your app.

To get ready for continuous deployment, choose which kind of deployment target you want, and then adjust your CI process as needed.

- [Azure web app or IIS server](#)
- [Linux VM](#)
- [Container](#)

All the tasks you need were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code. Proceed to finish the CI process definition.

## Finish the CI process definition

You're nearly ready to go. Just a few more steps to complete your CI build process.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

### 1. For the **Agent queue**:

- **VSTS:** Select *Hosted VS2017*. This is how you can use our pool of agents that have the software you need to build your app.
- **TFS:** Select a queue that includes a [Windows build agent](#).

### 2. Select **Get sources** and then:

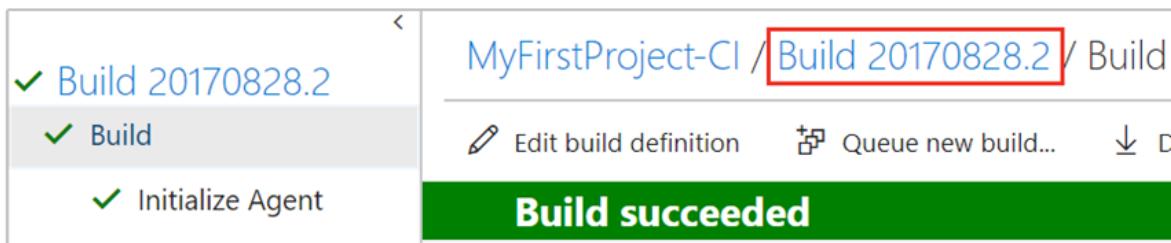
Observe that the new build definition is automatically linked to your repository.

### 3. Select the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.

4. Choose **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, choose **Save & queue**.
5. A new build is started. You'll see a link to the new build on the top of the page. Choose the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.



2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI build process in place to automatically build and validate whatever code is checked in by your team. What do you want to do next?

### Deploy your app

- [Azure web app or IIS server](#)
- [Linux VM](#)
- [Container](#)

#### IMPORTANT

Make sure you followed the **deployment instructions above** with the **Azure web app or IIS server** tab selected.

See one of the following:

- [Deploy to Azure Web App](#)
- [Deploy to a Windows VM](#)

### Extend to other Git workflows

Now that you have a CI build process for your master branch, you can extend the process to work with other branches in your repository, or to validate all pull requests. See:

- [CI builds for Git in VSTS](#)
- [CI builds for GitHub](#)

# Build your ASP.NET 4 app

12/20/2017 • 4 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

ASP.NET is a mature web platform that provides all the services that you require to build enterprise-class server-based web applications using .NET on Windows. Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your ASP.NET app whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have a version of Visual Studio matching your development machine installed on the agent machine.

## Get the sample code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

```
https://github.com/adventureworks/aspnet4-sample
```

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

This quickstart works for apps targeting the .NET Framework 4 or newer. The sample app is a Visual Studio solution that has two projects: An ASP.NET Web Application project that targets .NET Framework 4.5, and a Unit Test project.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the TFS Code hub interface. At the top, there's a navigation bar with 'MyFirstProject' and dropdown menus for 'Dashboards', 'Code', '...', 'Search work items', and user profile. Below the navigation bar is a secondary menu with 'SampleApp' (with a dropdown for 'master'), 'Files' (which is selected and highlighted in blue), 'History', 'Branches', 'Tags', and 'Pull Requests'. On the right side of the header, there are icons for 'Clone' and a user profile. The main content area shows a breadcrumb path 'master / SampleApp / Type to find a file or folder...' and a prominent blue button at the bottom right labeled 'Set up build' with a small icon above it.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, click **ASP.NET**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS**: Select *Hosted VS2017*. This is how you can use our pool of agents that have the software you need to build your app.
- **TFS**: Select a queue that includes a [Windows build agent](#).

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
6. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
7. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows a build summary card. On the left, there's a sidebar with a tree view: a checked green box for 'Build 20170828.2', a checked green box for 'Build', and a checked green box for 'Initialize Agent'. To the right of the sidebar is the main content area. The title of the content area is 'MyFirstProject-CI / Build 20170828.2 / Build'. Below the title are two buttons: 'Edit build definition' and 'Queue new build...'. At the bottom of the content area is a large green bar with the text 'Build succeeded' in white. There are also some partially visible icons and text on the far right.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)
- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build your GCC C/C++ app

2/20/2018 • 4 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your C/C++ application whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process for a C/C++ application compiled with GCC/g++.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have GCC installed on the agent machine.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

<https://github.com/adventureworks/cpp-gpp-sample>

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.
  - [VSTS or TFS repo](#)
  - [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the VSTS (Visual Studio Team Services) interface. At the top, there's a navigation bar with 'MyFirstProject' selected. Below it, a secondary navigation bar has 'Files' selected. On the right side of this bar, there's a blue button labeled 'Set up build' with a red border around it.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

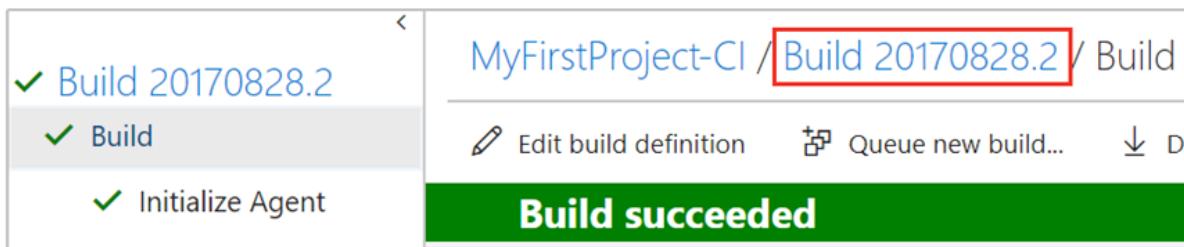
2. In the right panel, select **Empty**, and then click **Apply**. This template allows starting from scratch and adding your own build steps.
3. For the **Agent queue**:
  - **VSTS**: Select *Hosted Linux* or *Hosted macOS Preview*. This uses a VSTS pool of agents that have the software needed to build your app.
  - **TFS**: Select a queue that includes a [Linux](#) or [macOS](#) build agent.
4. Click **Get sources** and then:
  - [VSTS or TFS repo](#)
  - [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the + icon on **Phase 1** of the build and then:
    - [VSTS or TFS 2018](#)
    - [TFS 2017.2](#)
  1. Search for the **Shell Script** task and click **Add** to add it to your build.
  2. Click the **Shell Script** task and set its field values as follows:
- | FIELD   | VALUE                                 |
|---------|---------------------------------------|
| Version | 3.* or later                          |
| Type    | Inline                                |
| Script  | <code>g++ *.cpp -o hello-world</code> |
6. Click the **Triggers** tab and enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
  7. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
  8. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.



2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Publish your build output

Add the [Copy Files](#) and [Publish Build Artifacts](#) tasks to your build to save its compiled output as a build artifact.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)
- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build your Visual C++ app

2/16/2018 • 4 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your C/C++ application whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process for a C++ application compiled with Visual C++.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have a version of Visual Studio matching your development machine installed on the agent machine.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

<https://github.com/adventureworks/cpp-sample>

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

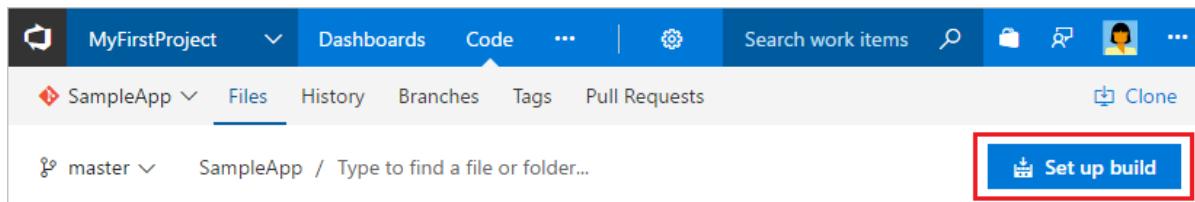
1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.
  - [VSTS or TFS repo](#)
  - [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.



You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, select **.NET Desktop**, and then click **Apply**. This template is useful in building most of the Visual Studio solutions including those that contain classic C++ projects.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS**: Select *Hosted VS2017*. This uses a VSTS pool of agents that have the software needed to build your app.
- **TFS**: Select a queue that includes a [Windows build agent](#).

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

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

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

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

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

7. Click the **Triggers** tab and enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.

8. Click the **Options** tab and then:

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

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

10. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.

11. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows the Azure DevOps build summary for a project named 'MyFirstProject-CI'. The build number is 'Build 20170828.2'. The status is 'Build succeeded'. There are links to 'Edit build definition' and 'Queue new build...'. A red box highlights the build number 'Build 20170828.2' in the top header.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)
- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build and push a Docker image

2/20/2018 • 4 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.3

This quickstart explains how to set up continuous integration (CI) to build a container image and push it to a Docker registry like Azure Container Registry or Docker Hub. At the end of this topic, you'll be ready to continuously deploy your image to a Kubernetes cluster or an Azure Web App for Containers.

## Prerequisites

- You must already have a CI build such as one of these:
  - [Build your ASP.NET Core app](#)
  - [Build your Go app](#)
  - [Build your Gradle app](#)
  - [Build your Node.js app with Gulp](#)
- An Azure subscription for pushing your container image to Azure Container Registry. If you don't have one, you can [create one for free](#).
- While the simplest way to try this quickstart is to use a VSTS account, you can also use TFS server. With TFS, make sure that you have [configured a build agent](#) with Docker installed.

## Create an Azure Container Registry

You can use [Azure Container Registry](#) to host the Docker image that is published by the CI process. Follow the steps below to create and configure a registry. In later steps, you use VSTS to deploy the image to an Azure Web App for Containers.

1. Sign into your Azure Account 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**.

## Create container registry

\* Registry name  
.azurecr.io

\* Subscription

\* Resource group i  
 Create new    Use existing

\* Location

\* Admin user i

---

\* Storage account

4. For **Admin user**, choose **Enable** and then choose **Create**.

5. Wait for the Azure Container Registry deployment to finish.

## Web or config as code

Do you want to define your build process in your web browser or configure it as code in YAML?

- [Web](#)
- [YAML](#)

### VSTS | TFS

Choose this option if you prefer a graphical interface in your web browser.



## Adapt your CI pipeline

Here you'll adapt your CI pipeline so that it builds and pushes your container image.

- [Web](#)
- [YAML](#)

### VSTS | TFS

Navigate to the **Builds** tab of the **Build and Release** hub in VSTS or TFS, and then edit your build pipeline. Select **Tasks**, and then add the following tasks:

 Docker	<ul style="list-style-type: none"><li>• <b>Azure subscription:</b> Select a connection from the list under <b>Available Azure Service Connections</b> or create a more restricted permissions connection to your Azure subscription. If you are using VSTS and if you see an <b>Authorize</b> button next to the input, click on it to authorize VSTS 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 <a href="#">Azure Resource Manager service endpoint</a> to manually set up the connection.</li><li>• <b>Azure Container Registry:</b> Select the Azure container registry that you created above.</li><li>• <b>Action:</b> Build an image.</li></ul>
 Docker	<ul style="list-style-type: none"><li>• <b>Azure subscription:</b> Same as in the previous task.</li><li>• <b>Azure Container Registry:</b> Same as in the previous task.</li><li>• <b>Action:</b> Push an image.</li></ul>

### Save and queue the build.

A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens. Verify that the Docker container image is built and pushed to your container registry.

Now your CI process is set up to push a new Docker image to a container registry every time a change is pushed to your application code.

## Next step

[Deploy to Azure Web App for Containers](#)

# Build your Go app

2/20/2018 • 4 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your Go application whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process for a Go application.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have Go installed on the agent machine.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

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

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

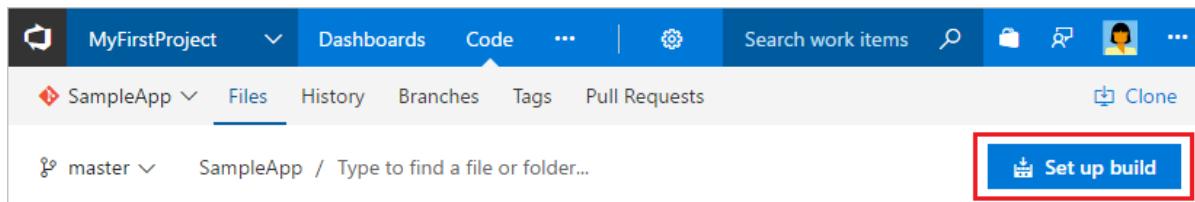
1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.
  - [VSTS or TFS repo](#)
  - [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.



The screenshot shows the VSTS interface for a project named 'MyFirstProject'. In the top navigation bar, 'Dashboards' and 'Code' are visible. Below the navigation, there's a menu for 'SampleApp' with options like 'Files', 'History', 'Branches', 'Tags', and 'Pull Requests'. On the right side of the header, there are icons for 'Search work items', 'Clone', and a user profile. A red box highlights the 'Set up build' button in the bottom right corner of the main content area.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, select **Empty**, and then click **Apply**. This template allows starting from scratch and adding your own build steps.
3. For the **Agent queue**:
  - **VSTS**: Select *Hosted Linux* or *Hosted macOS Preview*. This uses a VSTS pool of agents that have the software needed to build your app.
  - **TFS**: Select a queue that includes a [Linux or macOS build agent](#).

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the + icon on **Phase 1** of the build and then:

- [VSTS or TFS 2018](#)
- [TFS 2017.2](#)

1. Search for the **Shell Script** task and click **Add** to add it to your build.

2. Click the **Shell Script** task and set its field values as follows:

FIELD	VALUE
Version	3.* or later
Type	Inline
Script	go build hello.go

6. Click the **Triggers** tab and enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
7. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
8. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows a VSTS build summary page. The top navigation bar includes 'MyFirstProject-CI / Build 20170828.2 / Build'. Below the navigation are three build steps: 'Build' (highlighted in grey), 'Edit build definition', 'Queue new build...', and 'Delete'. Underneath the steps is a green bar with the text 'Build succeeded'. On the left, there's a sidebar with a tree view showing 'Build 20170828.2' expanded, with 'Build' and 'Initialize Agent' listed under it.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Publish your build output

Add the [Copy Files](#) and [Publish Build Artifacts](#) tasks to your build to save its compiled output as a build artifact.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)
- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build your Java app with Gradle

1/17/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your Java application whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

```
https://github.com/Adventworks/java-sample
```

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

The sample app in this repository is a Java servlet using JavaServer Pages (JSP). Tests for the application are written using JUnit. A Gradle wrapper file is used to build, test, and package the application into a web archive (.war) file.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the Azure DevOps interface. At the top, there's a navigation bar with 'MyFirstProject' and other options like 'Dashboards', 'Code', 'Search work items', and user profile. Below this is a secondary navigation bar for 'SampleApp' with tabs for 'Files', 'History', 'Branches', 'Tags', and 'Pull Requests'. A 'Clone' button is also present. The main content area shows a repository tree for 'master' branch under 'SampleApp'. On the right side, there's a prominent blue button labeled 'Set up build' with a gear icon, which is highlighted with a red box.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, search for `java`, select **Gradle**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS:** Select *Hosted Linux*, *Hosted macOS Preview*, or *Hosted VS2017*. This will use a hosted agent with the Java Development Kit (JDK) installed.
- **TFS:** Select a queue that includes an agent with the Java Development Kit (JDK) installed.

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the **Triggers** tab in the build definition. Enable the **Continuous integration** trigger. This will ensure that the build process is automatically triggered every time a change is committed to your repository.
6. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
7. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows the build summary for 'Build 20170828.2'. On the left, a sidebar lists the build steps: 'Build' (selected) and 'Initialize Agent'. The main area shows the build details: 'MyFirstProject-CL / Build 20170828.2 / Build'. Below this are links to 'Edit build definition' and 'Queue new build...'. A large green banner at the bottom states 'Build succeeded'.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)

- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build your Java app with Maven

1/17/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your Java application whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

```
https://github.com/Adventworks/java-sample
```

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

The sample app in this repository is a Java servlet using JavaServer Pages (JSP). Tests for the application are written using JUnit. A Maven POM file is used to build, test, and package the application into a web archive (.war) file.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the Azure DevOps interface. At the top, there's a navigation bar with 'MyFirstProject' and other options like 'Dashboards', 'Code', and 'Search work items'. Below the navigation is a header for 'SampleApp' with tabs for 'Files', 'History', 'Branches', 'Tags', and 'Pull Requests'. On the right side of the header, there's a 'Clone' button and a 'Set up build' button, which is highlighted with a red box. The main content area shows a file tree for 'master' with 'SampleApp' selected, and a search bar below it.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, search for `java`, select **Maven**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS:** Select *Hosted Linux*, *Hosted macOS Preview*, or *Hosted VS2017*. This will use a hosted agent with the Java Development Kit (JDK) installed.
- **TFS:** Select a queue that includes an agent with the Java Development Kit (JDK) and Maven installed.

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the **Triggers** tab in the build definition. Enable the **Continuous integration** trigger. This will ensure that the build process is automatically triggered every time a change is committed to your repository.
6. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
7. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows the build summary for 'Build 20170828.2'. On the left, there's a sidebar with a tree view showing 'Build 20170828.2' expanded, with 'Build' and 'Initialize Agent' nodes under it. On the right, the main pane shows the build details with the URL 'MyFirstProject-Cl / Build 20170828.2 / Build'. Below the URL, there are buttons for 'Edit build definition' and 'Queue new build...'. At the bottom, a large green bar displays the status 'Build succeeded'.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)

- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build your .NET desktop app for Windows

12/20/2017 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build your .NET desktop app whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have a version of Visual Studio matching your development machine installed on the agent machine.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

<https://github.com/adventureworks/net-sample>

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

This quickstart works for apps targeting the .NET Framework 4 or newer. The sample app is a Visual Studio solution that has two projects: A .NET Class Library project targeting .NET Framework 4.5 and a Unit Test project.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the Azure DevOps interface. At the top, there's a navigation bar with 'MyFirstProject' selected. Below it, a menu bar includes 'Dashboards', 'Code', '...', 'Search work items', and user icons. Under 'Code', there are links for 'SampleApp', 'Files', 'History', 'Branches', 'Tags', and 'Pull Requests'. A 'Clone' button is also present. In the main content area, a breadcrumb trail shows 'master / SampleApp / Type to find a file or folder...'. On the right side, a blue button labeled 'Set up build' is highlighted with a red border.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, select **.NET Desktop**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS**: Select *Hosted VS2017*. This is how you can use our pool of agents that have the software you need to build your app.
- **TFS**: Select a queue that includes a [Windows build agent](#).

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
6. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
7. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows the build summary for 'Build 20170828.2'. The left sidebar lists the build steps: 'Build' (selected) and 'Initialize Agent'. The main area shows the build details: 'MyFirstProject-CI / Build 20170828.2 / Build'. Below this are buttons for 'Edit build definition' and 'Queue new build...'. A large green banner at the bottom states 'Build succeeded'.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI process in place to automatically build and validate whatever code is checked in by your team. You can also automatically deploy your app. To learn more, see one of these topics:

- [Deploy to Azure Web App](#)

- [Deploy to a Linux VM](#)
- [Deploy to a Windows VM](#)

You can also modify this build definition to meet the needs of your team. To learn more see one of these topics:

- [CI builds for Git in VSTS](#)

# Build your Node.js app with Gulp

2/20/2018 • 10 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.3

Follow these steps to set up a continuous integration (CI) process for a Node.js app using Visual Studio Team Services (VSTS) or Team Foundation Server (TFS).

As you walk through this quickstart, we'll ask you to choose:

- Which kind of Git service you're using: VSTS or TFS, or GitHub.
- How you want to define your build: in a web interface, or configured as code in YAML
- For continuous deployment, what is your target: Azure web app or IIS server in a Windows VM, a Linux VM, or a Docker container.

As you choose from these options in the sections below, this topic will adapt to your choices.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have Node and Gulp installed on the agent machine.

## Get the sample code

The sample app we use here is a Node server that echoes "Hello world". Tests for the app are written using mocha framework. A gulp file is used to run the tests and to convert the results into junit format so that they can be published to VSTS or TFS.

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

```
https://github.com/adventworks/nodejs-sample
```

Where do you want to keep your code? Whichever service you choose, our system can automatically clone and pull code from it every time you push a change.

- [VSTS or TFS Git repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

You can also use Team Foundation Version Control (TFVC), Subversion, Bitbucket, or any other Git

repository for managing your source code, and still use VSTS for CI.

## Web or config as code

Do you want to define your build process in your web browser or configure it as code in YAML?

- [Web](#)
- [YAML](#)

### VSTS | TFS

Choose this option if you prefer a graphical interface in your web browser.



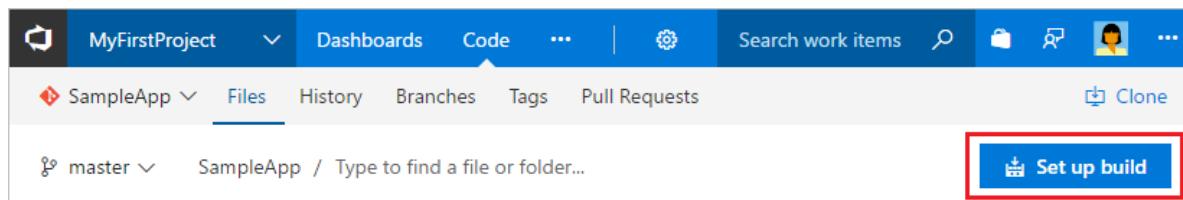
## Create the CI process definition

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

Begin by creating your build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

1. Navigate to the **Files** tab of the **Code** hub, and then choose **Set up build**.



You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, search for `node`, select **NodeJS with Gulp**, and then click **Apply**.

The screenshot shows the 'Build & Release' tab selected in the top navigation bar of the Azure DevOps interface. A search bar at the top right contains the text 'node'. Below it, a search result for 'NodeJS With Gulp (PREVIEW)' is shown, with a red box highlighting the 'Apply' button.

Select a template  
Or start with an [Empty process](#)

Others

**NodeJS With Gulp (PREVIEW)**  
Build NodeJS applications using Gulp task runner

**NodeJS With Grunt (PREVIEW)**  
Build NodeJS applications using Grunt task runner

Choose a template

Choose a template that builds your kind of app.  
Don't worry if it's not an exact match;  
you can add and customize the tasks later.

## Choose your deployment target

While a CI build process is a powerful way to do day-to-day development, continuous deployment is how many teams accelerate how they deliver value to customers. After each successful CI build, you can automatically deploy your app.

To get ready for continuous deployment, choose which kind of deployment target you want, and then adjust your CI process as needed.

- [Azure web app or IIS server](#)
- [Linux VM](#)
- [Container](#)

All the tasks you need were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

Select **Tasks**. Select the **Run gulp** task from the tasks. On the right side, you see the parameters for the task. Under the section JUnit Test Results, make sure the option to **Publish to TFS/VSTS** is selected.

Proceed to finish the CI process definition.

## Finish the CI process definition

You're nearly ready to go. Just a few more steps to complete your CI build process.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

### 1. For the **Agent queue**:

- **VSTS**: Select *Hosted Linux Preview*. This is how you can use our pool of agents that have the software you need to build your app.
- **TFS**: Select a queue that includes a [Windows build agent](#).

### 2. Select **Get sources** and then:

Observe that the new build definition is automatically linked to your repository.

3. Select the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
4. Choose **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, choose **Save & queue**.
5. A new build is started. You'll see a link to the new build on the top of the page. Choose the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You've just put your own CI build process in place to automatically build and validate whatever code is checked in by your team. What do you want to do next?

### Deploy your app

- [Azure web app or IIS server](#)
- [Linux VM](#)
- [Container](#)

#### IMPORTANT

Make sure you followed the **deployment instructions above** with the **Azure web app or IIS server** tab selected.

See one of the following:

- [Deploy to Azure Web App](#)
- [Deploy to a Windows VM](#)

### Extend to other Git workflows

Now that you have a CI build process for your master branch, you can extend the process to work with other branches in your repository, or to validate all pull requests. See:

- [CI builds for Git in VSTS](#)
- [CI builds for GitHub](#)

# Build your Universal Windows Platform app

12/20/2017 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Universal Windows Platform (UWP) is a common app platform available on every device that runs Windows 10. Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build and package your UWP app whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account. Make sure that you have [configured a build agent](#) for your team project, and that you have a version of Visual Studio matching your development machine installed on the agent machine.

## Get sample app code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

```
https://github.com/Microsoft/UWPQuickStart
```

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.

The screenshot shows the VSTS interface with the project 'MyFirstProject' selected. The top navigation bar includes 'Dashboards', 'Code', '...', 'Search work items', and user profile icons. Below the navigation is a secondary menu with 'SampleApp' (selected), 'Files' (highlighted with a blue underline), 'History', 'Branches', 'Tags', and 'Pull Requests'. A 'Clone' button is also present. The main content area shows a repository tree for 'master' branch under 'SampleApp' with a search bar 'Type to find a file or folder...'. A prominent blue button labeled 'Set up build' is highlighted with a red border.

You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, click **Universal Windows Platform**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS**: Select *Hosted VS2017*. This is how you can use our pool of agents that have the software you need to build your app.
- **TFS**: Select a queue that includes a [Windows build agent](#).

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Click the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
6. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
7. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows the build summary for 'Build 20170828.2'. The top navigation bar is visible with 'MyFirstProject' selected. The main content area shows the build status as 'Build succeeded'. Below the status are buttons for 'Edit build definition' and 'Queue new build...'. A green bar at the bottom of the summary section displays the message 'Build succeeded'.

2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

You can now update the build definition to generate production builds.

- [Signing UWP package](#)
- [Associate package with the store](#)

# Build your Xamarin app

12/19/2017 • 6 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Xamarin enables you to develop a single solution and deploy it to Android, iOS, and Windows devices. Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build and package your Xamarin app whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead of a VSTS account.
- You will build the sample app for Android and iOS using two build definitions in this quickstart. If you use VSTS, you can use a hosted agent for both. If you use TFS, you will need a private agent to build Xamarin.Android and Xamarin.iOS. Xamarin.iOS requires an agent running on macOS. Set up a private agent and [install Xamarin](#) on the agent machine. The Xamarin version on your development machine and build agent machine must be at least 4.0.3 for Windows and 5.10.3 for macOS.

BUILD	HOSTED AGENTS	ON-PREMISES WINDOWS AGENT	ON-PREMISES MACOS OR LINUX AGENT
Xamarin.Android	Yes	Yes (with Xamarin installed)	Yes (with Xamarin installed)
Xamarin.iOS	Yes	No	Yes (with Xamarin installed)
UWP	Yes	Yes (Windows 10)	No

## Get the sample code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

<https://github.com/adventworks/xamarin-sample>

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

# Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

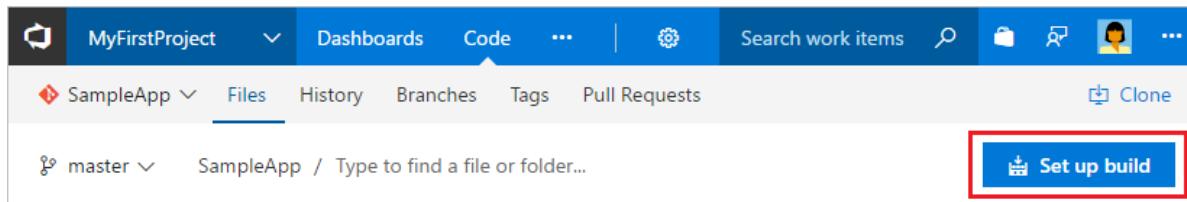
You need to create two build definitions - one for Xamarin.Android and one for Xamarin.iOS.

## Define your Xamarin.Android build

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.



You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, click **Xamarin.Android**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**:

- **VSTS:** Select *Hosted VS2017*. This hosted pool of agents has the software needed to build your app.
- **TFS:** Select a queue that includes a [macOS](#) or [Windows](#) build agent.

4. Click **Get sources** and then:

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

5. Select the **Build Xamarin.Android Project** task. In the properties for this task, select **JDK 8** as the **JDK Version**, and **x64** as the **JDK Architecture**.

6. Select the **Build solution \*/test.csproj** task. In the properties for this task, uncheck **Enabled** under **Control Options**. There are no tests in the sample repository.

7. Select the **Xamarin Test Cloud** task. Remove this task from the definition by right-clicking it and selecting **Remove selected task(s)**.

8. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.

9. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

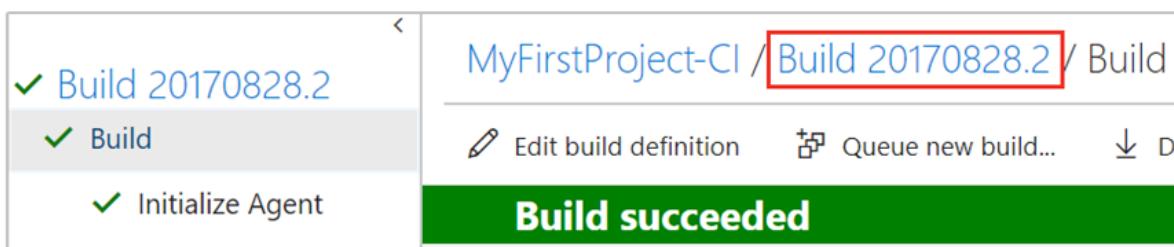
## Define your Xamarin.iOS build

Navigate to the **Builds** tab of the **Build and Release** hub, and then click **+ New**. You are asked to **Select a template** for the new build definition. This time, select the **Xamarin.iOS** template.

1. For the **Agent queue**, select a hosted macOS queue such as **Hosted macOS Preview**, or the private queue that includes your macOS agent.
2. For the **Solution to build**, enter `HelloXamarinFormsWorld.sln`.
3. Remove the **Xamarin Test Cloud** task by right-clicking it and selecting **Remove selected task(s)**.
4. Select the **Build Xamarin.iOS solution** task. In the properties for this task, enable the **Build for iOS Simulator** check box.
5. Click **Save & queue** to kick off the Xamarin.iOS build. On the **Save build definition and queue** dialog box, click **Save & queue**.
6. A new build is started. You will see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.



2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

To be able to configure your own app for iOS release, you need to make the following changes in the solution in your development environment, since the Xamarin.iOS build requires a solution configuration that builds only the Xamarin.iOS project and its dependencies.

1. In Visual Studio, open **Solution Explorer** (Keyboard: Ctrl + Alt + L).
2. Right-click your solution and then click **Configuration Manager**.
3. On the configuration manager dialog box open the active solution configuration drop-down menu and click **New**.
4. On the new solution configuration dialog box:
  - For Name, enter `iOS Release`
  - Open **Copy settings from** drop-down menu and select **Release**.
  - Clear the **Create new project configurations** dialog box.
5. Open the **Active solution platform** drop-down menu:
  - a. Select **iPhoneSimulator** and clear the check boxes on all rows except your Xamarin.iOS project and any projects (for example, portable class libraries) it depends on.
  - b. Repeat this step for **iPhone**.

6. File -> Save All (Keyboard: Ctrl + Shift + S).

7. Check in your changes.

There's also a known issue that might cause a problem with building your Xamarin.iOS project. For example, in the build log for a Xamarin.iOS build step you might see an errors such as *error : Project reference 'App1/App1.csproj' has invalid or missing guid for metadata 'Project'*.

To fix this issue:

1. In Visual Studio, open **Solution Explorer** (Keyboard: Ctrl + Alt + L).
2. Expand your .iOS project node, and then the **References** node.
3. Right-click each reference to a portable class library and then click **Remove**.
4. Right-click the **References** node and then click **Add Reference**.
5. On the **Reference Manager** dialog box, expand **Projects**, and then click **Solution**.
6. Select the portable class library projects you removed and click **OK**.
7. File -> Save All (Keyboard: Ctrl + Shift + S).
8. Check in your changes.

# Build your Xcode app

1/26/2018 • 5 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly customizable continuous integration (CI) process to automatically build and package your Xcode app whenever your team pushes or checks in code. In this quickstart you learn how to define your CI process.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- While the simplest way to try this quickstart is to use a VSTS account, you can also use a TFS server instead.
- First, you will need a build agent configured on a Mac machine. You may use one of the following:
  1. The **Hosted macOS Preview** agent provided by VSTS, or
  2. Provide your own agent by opening the macOS Terminal app on your Mac and following these [setup instructions](#). The agent will automatically register itself with VSTS / TFS when you start it for the first time. Your Mac also needs to have Nodejs, Xcode, and [xcpretty](#) (for testing) installed.

## Get the sample code

You can copy this sample app code directly into your version control system so that it can be accessed by your CI build process. To get started, copy this URL to your clipboard:

```
https://github.com/adventworks/xcode-sample
```

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

The sample provided here is an iOS app, but the concepts described here translate to other Xcode builds such as for macOS, tvOS, and watchOS apps. Results from running tests are published to VSTS using [xcpretty](#). That is why you will need to have xcpretty installed if you are using your own Mac machine to perform builds, since xcpretty is not part of Xcode itself.

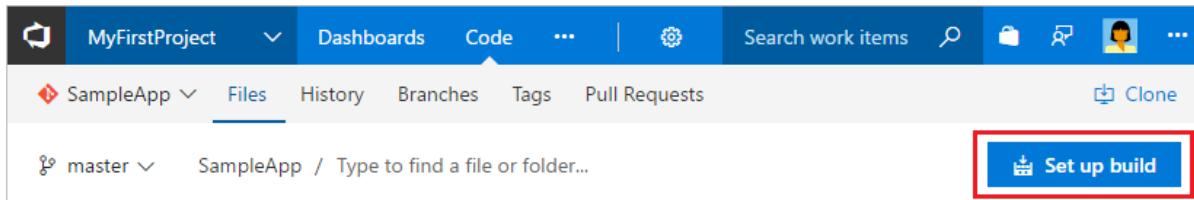
## Set up continuous integration

A continuous integration (CI) process automatically builds and tests code every time a team member commits changes to version control. Here you'll create a CI build definition that helps your team keep the master branch clean.

1. Create a new build definition.

- [VSTS or TFS repo](#)
- [GitHub repo](#)

Navigate to the **Files** tab of the **Code** hub, and then click **Set up build**.



You are taken to the **Build and Release** hub and asked to **Select a template** for the new build definition.

2. In the right panel, click **Xcode**, and then click **Apply**.

You now see all the tasks that were automatically added to the build definition by the template. These are the steps that will automatically run every time you check in code.

3. For the **Agent queue**, select **Hosted macOS Preview** or a queue that includes the Mac agent you set up.
4. For the **Scheme**, enter `iOSHelloWorld`
5. Make sure that each of the Xcode steps are set to use version **4.\*** or later.
6. Click **Get sources** and then:
  - [VSTS or TFS repo](#)
  - [GitHub repo](#)

Observe that the new build definition is automatically linked to your repository.

7. Click the **Triggers** tab in the build definition. Enable the **Continuous Integration** trigger. This will ensure that the build process is automatically triggered every time you commit a change to your repository.
8. Click **Save & queue** to kick off your first build. On the **Save build definition and queue** dialog box, click **Save & queue**.
9. A new build is started. You'll see a link to the new build on the top of the page. Click the link to watch the new build as it happens.

## Troubleshooting tips

If you encounter a "User interaction not allowed" error when running the agent as a launch agent, on the **Xcode** task, you will either need enable the "Unlock default keychain" option, or switch to referencing signing certificates using a file. See [Sign your mobile app](#) for details.

If you run into issues with your tests hanging and/or not being able to start the iOS Simulator at times, you can add the **Command Line** task to run the `killall` tool with "Simulator" as an argument (i.e. `killall "Simulator"` ). This will force the simulator to shut down in the event it is hung. Exercise care when running the command if you have multiple agents running for the same user and that you do not accidentally kill other processes.

## View the build summary

1. Once the build completes, select the build number to view a summary of the build.

The screenshot shows a build summary for 'MyFirstProject-CI / Build 20170828.2 / Build'. On the left, there's a sidebar with a tree view showing a checked 'Build 20170828.2' node, a 'Build' node (which is selected and highlighted in grey), and an 'Initialize Agent' node. On the right, the main area has a header with 'Edit build definition' and 'Queue new build...'. Below the header is a large green bar with the text 'Build succeeded' in white. The entire interface is set against a light grey background.

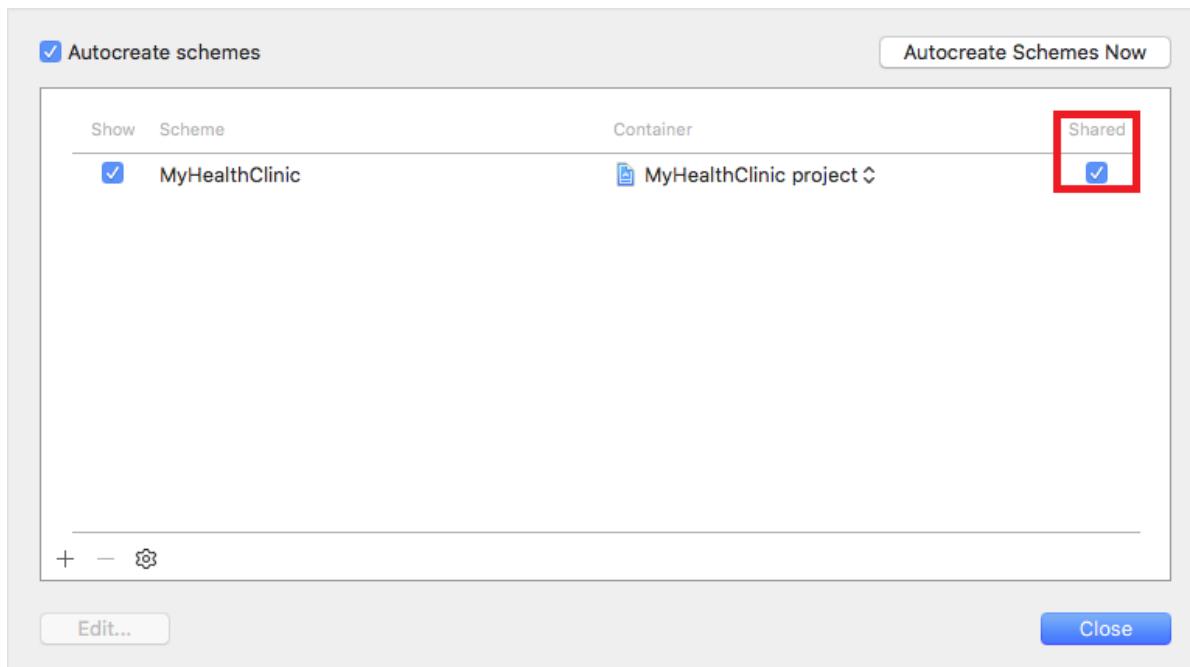
2. Notice the various sections in the build summary - the source version of the commit in build details section, list of all associated changes, links to work items associated with commits, and test results. When the build is automatically triggered by a push to your Git repository, these sections are populated with all the relevant information.

## Next steps

To sign your application with a certificate and provisioning profile as part of CI, see [Sign your mobile app](#).

If you plan to use your own Xcode project for this quickstart, an additional step is required to configure your project for a CI environment. Mark a scheme of your Xcode project as "Shared" and add it to source control to be used during your CI builds. Follow these steps:

1. In Xcode, open your project and go to **Product > Scheme > Manage Schemes...**
2. Enable **Shared** next to the scheme you want to use during CI. Remember the name of the scheme you shared as we will reference it later.
3. Now add the new files and folders in your .xcodeproj folder (specifically the xcshareddata folder to source control).



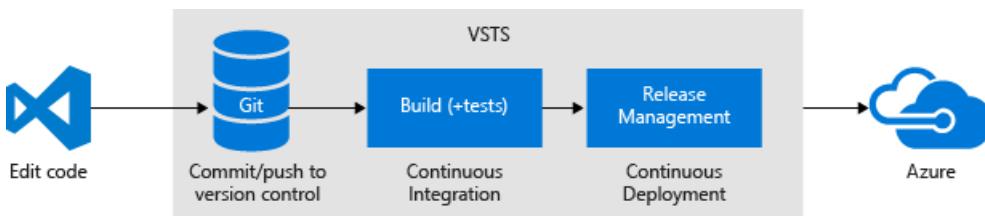
# Build and deploy to an Azure Web App

2/26/2018 • 4 min to read • [Edit Online](#)

## VSTS

Visual Studio Team Services (VSTS) provides a highly customizable continuous integration (CI) and continuous deployment (CD) pipeline to automatically deliver your ASP.NET Core web app to Azure App Services. In this quickstart you will use the Azure portal to configure an entire CI/CD pipeline, and then see it build and deploy your app.

For example, you can continuously deliver your app to Azure.



After you commit and push a code change, it is automatically built and then deployed. The results will automatically show up on your site.

## Prerequisites

- A VSTS account. 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 team project you want to use.
- An Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).

## Get the sample code

To configure a CI build process for your app, the source code needs to be in a version control system. You can use VSTS as your version control system, since VSTS provides a full-featured Git server. Alternatively, you can use GitHub.

For a simple way to follow this quickstart, get the following sample app code and put it into your own version control repository:

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

- [VSTS or TFS repo](#)
- [GitHub repo](#)

To import the sample app into a Git repo in VSTS or TFS:

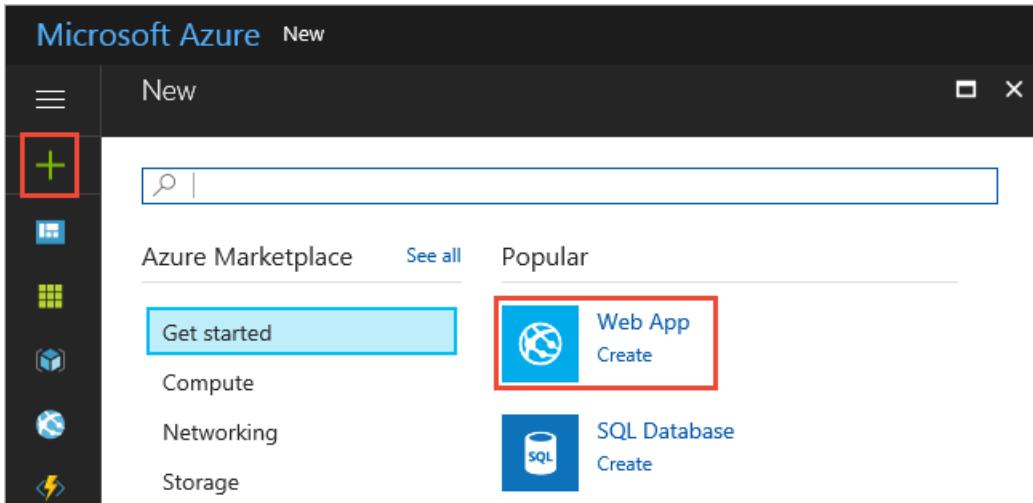
1. On the **Code** hub for your team project in VSTS/TFS, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the above URL into the **Clone URL** text box.
3. Click **Import** to copy the sample code into your Git repo.

## Create an Azure web app using the portal

**NOTE**

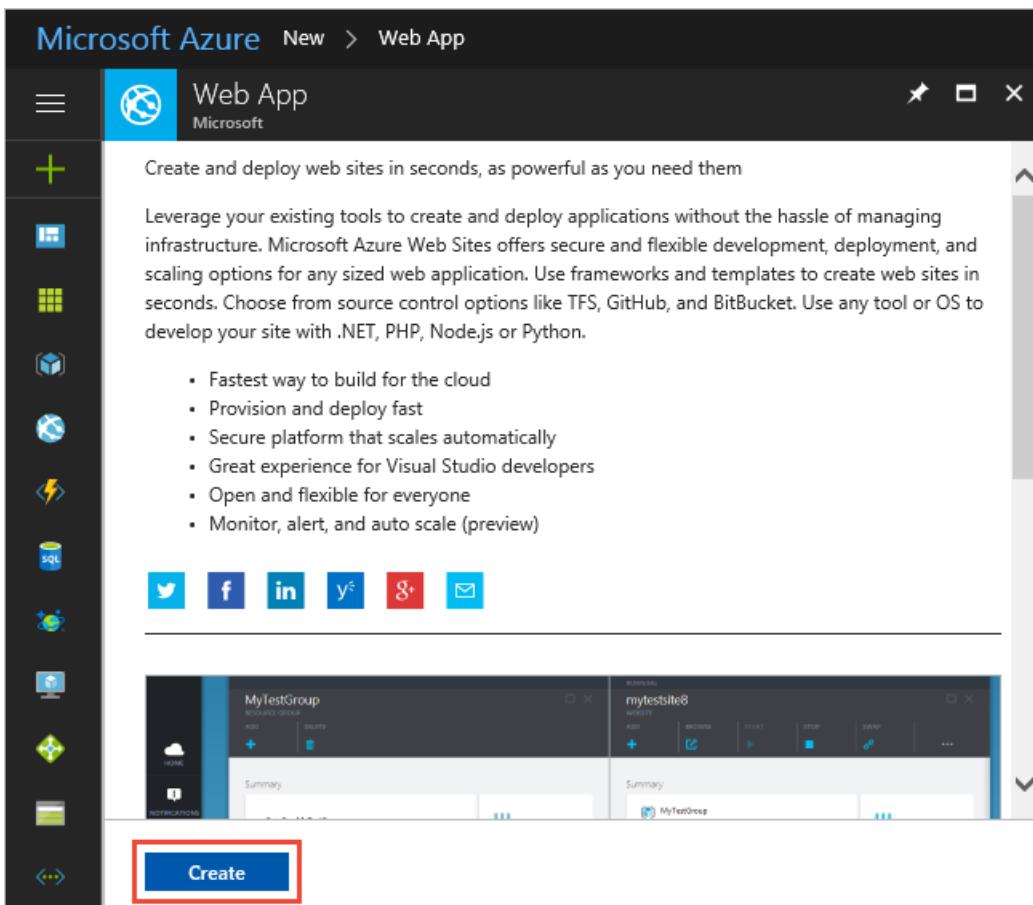
If you already have a web app that you want to use, you can skip this and move to the next section.

1. Sign into the [Microsoft Azure portal](#).
2. Choose the + icon in the left navigation bar, then choose **Web App**.

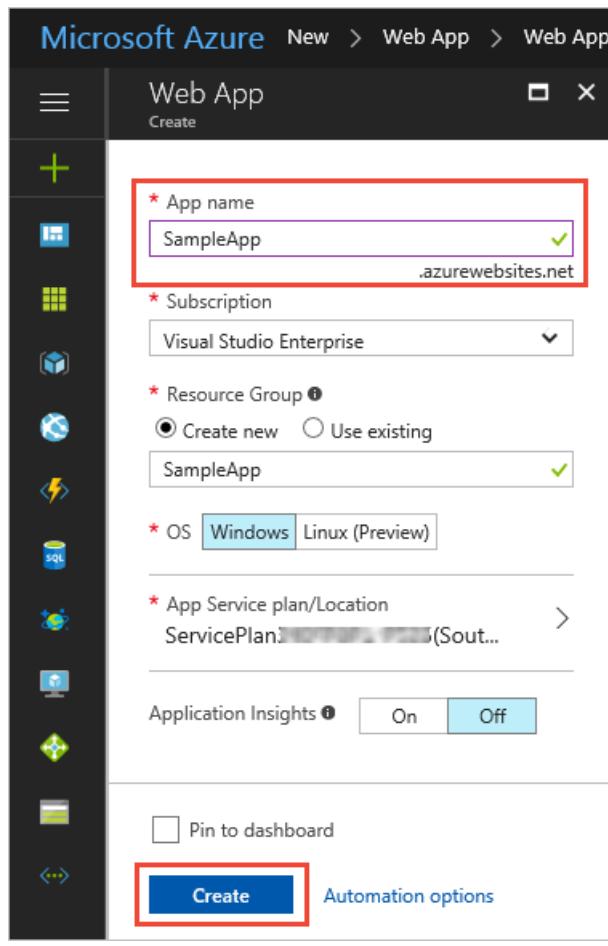


If you don't see **Web App** in the list, use the search box to find it.

3. At the bottom of the introduction page, choose **Create**.



4. Enter a name for the new web app. You'll see a green checkmark when the name is unique. Then choose **Create**.



5. Choose the **App Services** icon in the left navigation bar. After a few minutes, you'll see the new web app appear in the list. Ensure that it is running.

App Services	
<a href="#"></a>	<a href="#">Add</a>
<a href="#"></a> <a href="#">Assign Tags</a> <a href="#"></a> <a href="#">Columns</a> <a href="#"></a> <a href="#">Refresh</a> <a href="#"></a> <a href="#">Start</a> <a href="#"></a> <a href="#">Restart</a>	
1 items	
<a href="#"></a>	<a href="#">NAME</a> <a href="#">STATUS</a> <a href="#">APP TYPE</a> <a href="#">APP SERVICE...</a> <a href="#">LOCATION...</a>
<a href="#"></a>	<a href="#">SampleApp</a> <a href="#">Running</a> <a href="#">Web app</a> <a href="#">ServicePlan...</a> <a href="#">South Cen...</a>

You're now ready to start using your new web app.

## Configure continuous delivery

1. In the Azure portal, open your web app's blade. Choose **Continuous Delivery** and then choose **Configure**.

The screenshot shows the Microsoft Azure portal interface for a sample application named 'SampleApp'. In the left sidebar, there's a list of resources including 'myAppServicePlan' and 'SampleApp'. The main area is titled 'SampleApp - Continuous Delivery (Preview)' and contains sections for 'Deploy with confidence' (with four green checkmarks) and a deployment pipeline diagram. A 'Configure' button is highlighted with a red box.

2. Select **Choose repository** and select **VSTS** for the code repository. Select the project, repository, and branch into which your imported the sample code. When you're done, choose **OK**.

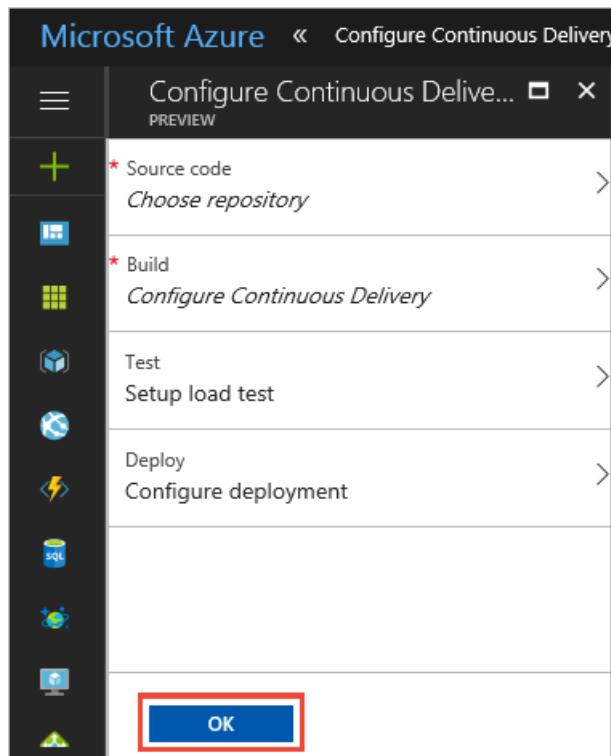
This screenshot shows the 'Choose repository' configuration dialog. It includes fields for 'Code repository' (set to 'Visual Studio Team Services'), 'Visual Studio Team Services account' (set to 'dotnetcore'), 'Project' (set to 'MyFirstProject'), 'Repository' (set to 'dotnetcore-sample'), and 'Branch' (set to 'master'). At the bottom, there are two 'OK' buttons; the right one is highlighted with a red box.

3. Select **Configure Continuous Delivery** and choose **ASP.NET Core**. When you're done, choose **OK**.

This screenshot shows the 'Configure Continuous Delivery' configuration dialog. It includes a dropdown for 'Web Application framework' (set to 'ASP.NET Core'). At the bottom, there is an 'OK' button highlighted with a red box.

4. Skip the other two steps - **Test** and **Deploy** - and choose **OK** to complete the configuration of continuous

delivery. You'll see how to use the test and deployment options in other tutorials.



- When you choose **OK**, Azure Continuous Delivery configures and kicks off a build and deployment in VSTS. When the build completes, the deployment is automatically initiated. After a while, the deployment is completed. Choose **Refresh Logs** to see this in the **Activity Log**.

A screenshot of the Azure Activity Log. The top navigation bar includes 'Refresh Logs' (which is highlighted with a red box), 'Disconnect', 'Edit', and 'Sync'. Below the navigation is a summary for 'Production': 'Build 51' and 'Release 1'. A flow diagram shows 'Code' leading to 'Build' leading to 'Deploy'. The 'Activity logs' section for 7/21/2017 shows two entries:

- Deployed successfully to Production (Source Version d8bacc3e4f, Build 51, Release 1) at 12:26 PM
- Successfully setup Continuous Delivery and triggered build (Build Definition, Release Definition, Build triggered) at 12:22 PM

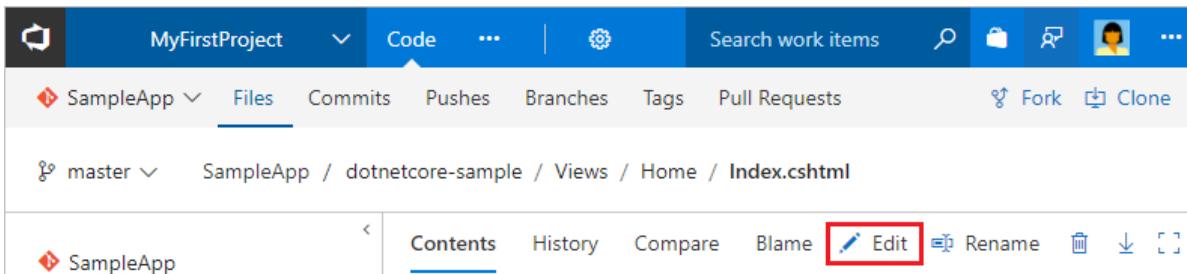
- Open a new browser window and navigate to your new web at <http://your-app-name.azurewebsites.net>.

## Edit the code and see it deployed

Now that you have a completely automated CI/CD pipeline, any changes that you make to the app are automatically built and deployed by VSTS. To try this, make a change to the app and commit that change to the Git repository.

- Both VSTS and GitHub feature a full code editor within the web browser. Using the browser, navigate to **Views/Homes/Index.cshtml** file in your repository.
  - VSTS or TFS repo
  - GitHub repo

In the VSTS **Code** hub, edit the **Views/Home/Index.cshtml** file.



The screenshot shows the VSTS Code hub interface. At the top, there's a navigation bar with 'MyFirstProject' and 'Code' selected. Below it is a header bar with 'SampleApp' dropdown, 'Files' (selected), 'Commits', 'Pushes', 'Branches', 'Tags', 'Pull Requests', and user icons for 'Fork' and 'Clone'. The main area shows a file tree: master > SampleApp / dotnetcore-sample / Views / Home / Index.cshtml. In the bottom right of this tree view, there are buttons for 'Contents', 'History', 'Compare', 'Blame', 'Edit' (which is highlighted with a red box), 'Rename', and download options. The code editor below shows the following HTML:

```
<h1>Demo of CI/CD!!</h1>
```

2. Make a simple change above the slide carousel `div` tag:

```
<h1>Demo of CI/CD!!</h1>
```

3. **Commit** your changes to trigger a CI build. When the build completes, it triggers an automatic deployment.

It takes several minutes for the build and deployment to execute. Wait until the deployment is done, then verify that your changes are live in your web browser: <http://your-app-name.azurewebsites.net>.

You're now ready to collaborate with a team on an ASP.NET Core app with a CI/CD process that automatically deploys your latest work to your web site.

## Next steps

When you configured your CI/CD process in this quickstart, a build and release definition were automatically created in your VSTS project. You can modify these build and release definitions to meet the needs of your team. To learn more see one of these tutorials:

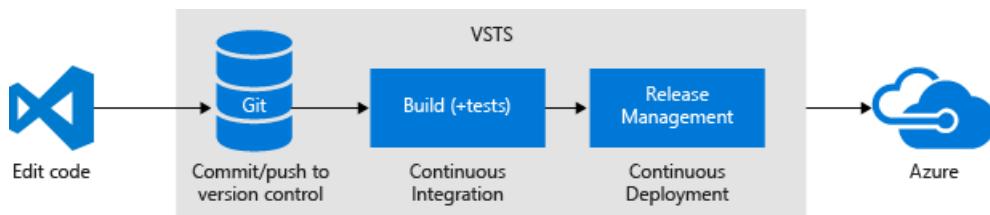
- [Customize CD process](#)

# Deploy to a Windows Virtual Machine

2/26/2018 • 4 min to read • [Edit Online](#)

## VSTS | TFS 2018

We'll show you how to set up continuous deployment of your ASP.NET or Node app to an IIS web server running on Windows using Visual Studio Team Services (VSTS). You can use the steps in this quickstart as long as your continuous integration process 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 process

You'll need a continuous integration (CI) build process that publishes your web deployment package. To set up a CI build process, see:

- [Build your ASP.NET 4 app](#)
- [Build your ASP.NET Core app](#)
- [Build your Node app with Gulp](#)

## 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-x64.1.0.4.exe
Start-Process $env:temp\dotnet-dev-win-x64.1.0.4.exe -ArgumentList '/quiet' -Wait

# Install the .NET Core Windows Server Hosting bundle
Invoke-WebRequest https://go.microsoft.com/fwlink/?LinkId=817246 -outfile
$env:temp\DotNetCore.WindowsHosting.exe
Start-Process $env:temp\DotNetCore.WindowsHosting.exe -ArgumentList '/quiet' -Wait

# Restart the web server so that system PATH updates take effect
net stop was /y
net start w3svc

```

When `net start w3svc` appears, press **Enter** to run it.

#### Node app

Follow the instructions in [this topic](#) to install and configure IISnode on IIS servers.

## Create a deployment group

Deployment groups in VSTS 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 a VSTS agent on each of them. Each machine interacts with VSTS to coordinate deployment of your app.

1. Open the VSTS web portal (<https://{your-account}.visualstudio.com>), navigate to the **Build and Release** hub, and then click **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**.

Deployment Groups / myIIS

Details Machines Save Security Help

Deployment group name: myIIS

Register machine

Type of machine(s) to register: Windows

Registration script (PowerShell)

```
$ErrorActionPreference="Stop";If(-NOT ([Security.Principal.WindowsPrincipal] [Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole ([Security.Principal.WindowsBuiltInRole] "Administrator")){ throw "Run command in Administrator PowerShell Prompt"};If(-NOT (Test-Path $env:systemDrive'\vstsagent') ){mkdir $env:systemDrive'\vstsagent'}; cd $env:systemDrive'\vstsagent'; for($i=1; $i -lt 100; $i++) {${destFolder="A"+$i.ToString();}if(-NOT (Test-Path ${destFolder})) {mkdir ${destFolder};cd ${destFolder};break;}}; $agentZip="$PWD\agent.zip";(New-Object Net.WebClient).DownloadFile('https://github.com/Microsoft/vsts-agent/releases/download/v2.119.1/vsts-agent-win7-x64-2.119.1.zip', $agentZip);Add-type -AssemblyName System.IO.Compression.FileSystem; [System.IO.Compression.ZipFile]::ExtractToDirectory($agentZip, "$PWD");.\config.cmd --deploymentgroup --agent $env:COMPUTERNAME --runnasservice --work '_work' --url 'https://mvcdocs1.visualstudio.com/' --projectname 'MyFirstProject' --deploymentgroupname "myIIS" ; Remove-Item $agentZip;
```

Use a personal access token in the script for authentication

**Copy script to clipboard**

Run from Administrator PowerShell command prompt

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 of the **Build and Release** hub in VSTS, open the *myIIS* deployment group. On the **Machines** tab, verify that your VM is listed.

The screenshot shows the 'Deployment Groups' page in VSTS. The 'Targets' tab is selected. A single target, 'myVM', is listed as healthy and never deployed. It is associated with a 'Web' tag.

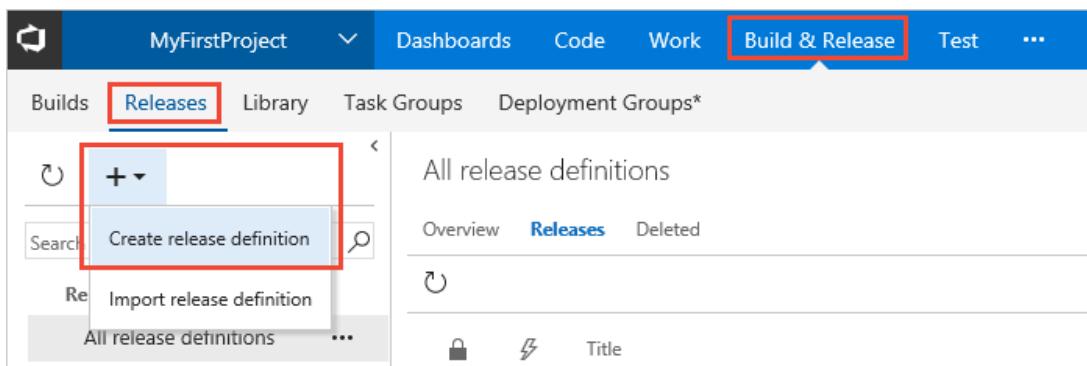
## Define your CD release process

Your CD release process picks up the artifacts published by your CI build and then deploys them to your IIS servers.

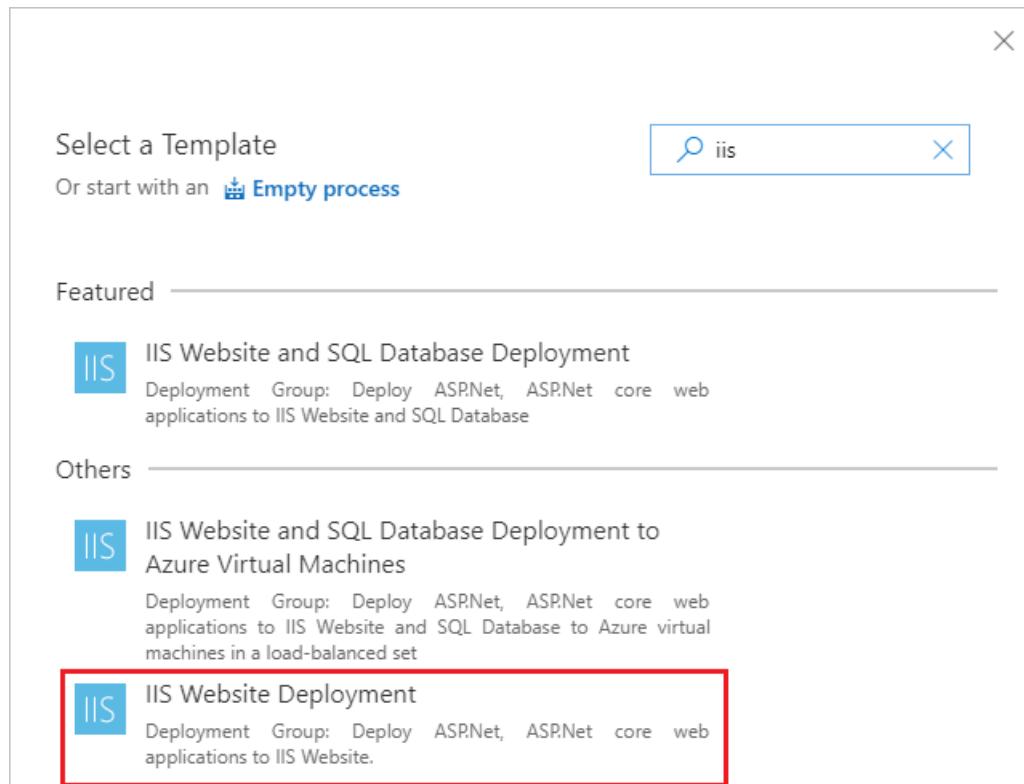
1. Do one of the following:
  - If you've just completed a CI build then, in the build's **Summary** tab under **Deployments**, choose **Create release** followed by **Yes**. This starts a new release definition that's automatically linked to the build definition.

The screenshot shows the 'Build succeeded' summary for 'SampleApp / Build 20170815.1'. The 'Release' button is highlighted with a red box.

- Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**.



2. In the **Create release definition** wizard, select **IIS Website Deployment** template, and then click **Apply**.



Select a Template

Or start with an [Empty process](#)

Featured

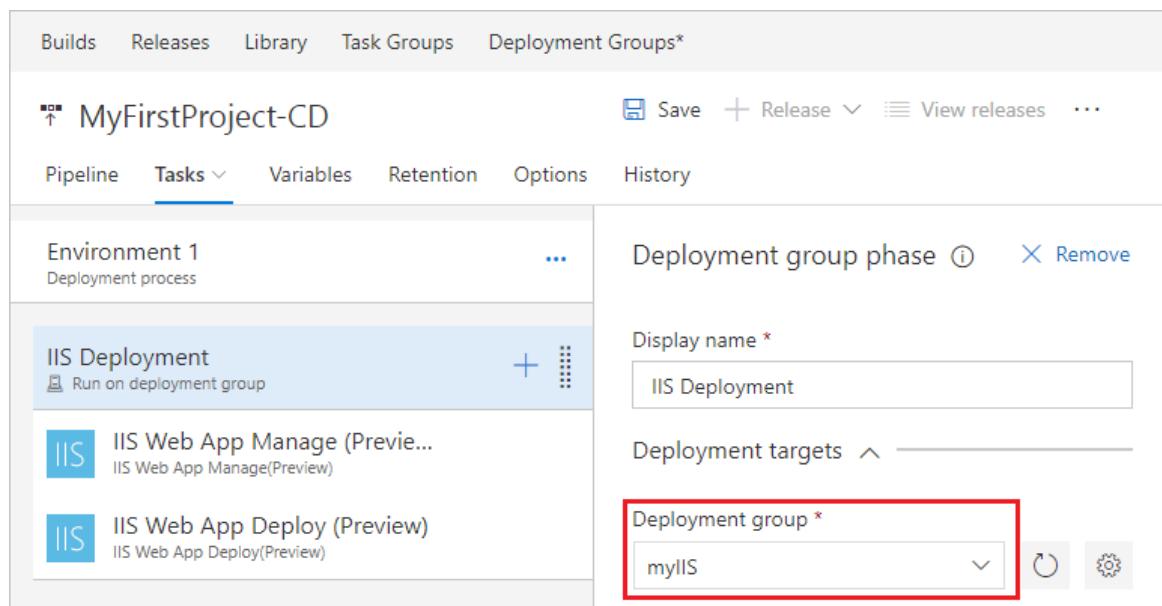
**IIS** IIS Website and SQL Database Deployment  
Deployment Group: Deploy ASPNet, ASP.Net core web applications to IIS Website and SQL Database

Others

**IIS** IIS Website and SQL Database Deployment to Azure Virtual Machines  
Deployment Group: Deploy ASPNet, ASP.Net core web applications to IIS Website and SQL Database to Azure virtual machines in a load-balanced set

**IIS** IIS Website Deployment  
Deployment Group: Deploy ASPNet, ASP.Net core web applications to IIS Website.

3. Click the **Tasks** tab, and then click the **IIS Deployment** phase. For the **Deployment Group**, click the deployment group you created earlier, such as *myIIS*.



Builds Releases Library Task Groups Deployment Groups\*

MyFirstProject-CD

Save + Release View releases ...

Pipeline **Tasks** Variables Retention Options History

Environment 1 Deployment process

**IIS Deployment** Run on deployment group

**IIS** IIS Web App Manage (Preview...) IIS Web App Manage(Preview)

**IIS** IIS Web App Deploy (Preview) IIS Web App Deploy(Preview)

Deployment group phase [Remove](#)

Display name \* IIS Deployment

Deployment targets

Deployment group \* myIIS

4. Save the release definition.

## 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 definition with the artifacts produced by a specific build. This will result in deploying the build:

1. To test the release definition, choose **Release** and then **Create release**.
2. In the Create new release panel, choose **Create**. Choose the link near the top of the window that indicates a new release was created.
3. Open the **Logs** tab to watch the live logs from the deployment as it happens. Wait for the release to be deployed to the Azure web app.
4. Once deployment has completed, open your web browser and test your web app:  
`http://<publicIpAddress>`, where `<publicIpAddress>` is the IP address of your web site on your IIS web server.

## Next steps

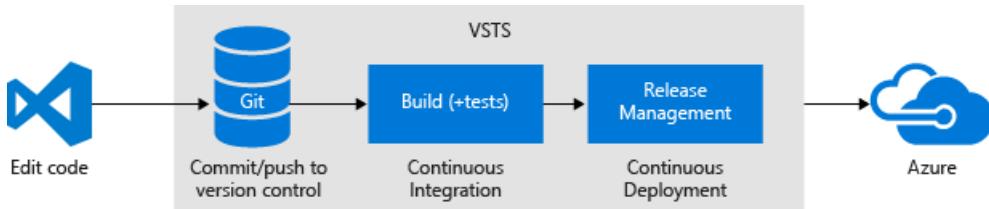
- [Dynamically create and remove a deployment group](#)
- [Apply environment-specific configurations](#)
- [Perform a safe rolling deployment](#)
- [Deploy a database with your app](#)

# Deploy to a Linux Virtual Machine

2/26/2018 • 5 min to read • [Edit Online](#)

## VSTS | TFS 2018

We'll show you how to set up continuous deployment of your app to an nginx web server running on Ubuntu using Visual Studio Team Services (VSTS) or Team Foundation Server (TFS) 2018. You can use the steps in this quickstart for any app as long as your continuous integration process 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 process

You'll need a continuous integration (CI) build process that publishes your web application, as well as a deployment script that can be run locally on the Ubuntu server. To set up a CI build process, see:

- [Build your Node app with Gulp](#)

Make sure you follow the additional steps in that topic for creating a build to deploy to Linux.

## Prerequisites

You'll need a Linux VM with Nginx web server to deploy the app. The deployment scripts used in the sample repositories have been tested on Ubuntu 16.04, and we recommend you use the same version of Linux VM for this quickstart. If you don't already have a Linux VM with Nginx, create one now in Azure using the steps in [this example](#).

## Create a deployment group

Deployment groups in VSTS make it easier to organize the servers you want to use to host your app. A deployment group is a collection of machines with a VSTS agent on each of them. Each machine interacts with VSTS to coordinate deployment of your app.

1. Open a SSH session to your Linux VM. You can do this using the Cloud Shell button on the menu in the upper-right of the [Azure portal](#).



2. Initiate the session by typing the following command, substituting the IP address of your VM:

```
ssh <publicIpAddress>
```

For more information, see [SSH into your VM](#).

3. Run the following command:

```
sudo apt-get install -y libunwind8 libcurl3
```

The libraries this command installs are pre-requisites for installing the build and release agent onto a Ubuntu 16.04 VM. Pre-requisites for other versions of Linux can be found [here](#).

4. Open the VSTS web portal <https://{your-account}.visualstudio.com>, navigate to the **Build & Release** hub, 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**.

Deployment Groups > myNginx

Details Targets Save Security Help

Deployment group name: myNginx

Type of target to register: Ubuntu 16.04+ (? System prerequisites)

Description:

Registration script:

```
mkdir vstsagent;cd vstsagent;curl -fksL -o vstsagent https://github.com/Microsoft/vsts-agent/releases/download/agent-ubuntu.16.04-x64-2.124.0.tar.gz;tar -zxf vstsagent.tar.gz;./config.sh --deploymentgroup --accept-hostname-self $HOSTNAME --url https://ahomer-ms.visualstudio.com/ --projectname 'MyFirstProject' --deploymentgroupname --runasservice;sudo ./svc.sh install;sudo ./svc.sh start
```

Use a personal access token in the script for authentication

**Copy script to clipboard** Run from Administrator command prompt

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 VSTS Agent*. Type "q" to exit the file editor and return to the shell prompt.
11. Back in VSTS or TFS, on the **Deployment groups** page of the **Build & Release** hub in VSTS, open the **myNginx** deployment group. On the **Targets** tab, verify that your VM is listed.

Deployment Groups > myNginx

Details Targets Save Security Help

✓ Healthy (1) Summary Tags

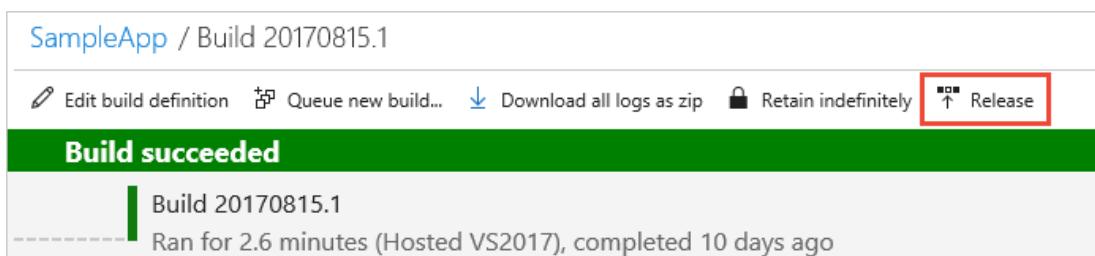
myNginxServer Never deployed Add tags

# Define your CD release process

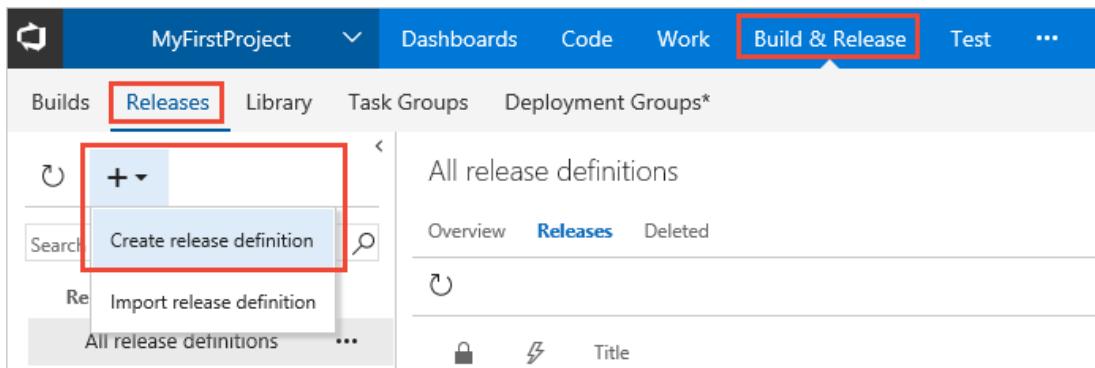
Your CD release process picks up the artifacts published by your CI build and then deploys them to your nginx servers.

1. Do one of the following to start creating a release definition:

- If you've just completed a CI build then, in the build's **Summary** tab under **Deployments**, choose **Create release** followed by **Yes**. This starts a new release definition that's automatically linked to the build definition.



- Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**.



2. In the **Create release definition** wizard, select **Empty process**.
3. If you created your new release definition from a build summary, check that the build definition and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release definition from the **Releases** tab, choose the **+ Add** link and select your build artifact.

The screenshot shows the 'Add artifact' dialog in the Azure DevOps interface. The 'Source type' section has 'Build' selected. The 'Project' field is set to 'MyFirstProject'. The 'Source (Build definition)' dropdown is set to 'MyFirstProject-NodeJS With Gulp (PREVIEW)-CI'. The 'Default version' is set to 'Latest'. A note below states: 'No version available for MyFirstProject-NodeJS With Gulp (PREVIEW)-CI version has no artifacts to publish. Please check the source definition.' The 'Add' button at the bottom is highlighted with a red box.

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and the **master** branch is selected. If not, set these options now.

The screenshot shows the 'Continuous deployment trigger' settings in the Azure DevOps pipeline. The 'Enabled' toggle switch is turned on. The 'Build branch' dropdown is set to 'master'. A note at the bottom states: 'Continuous deployment is not enabled by default when you create a new release definition from the Releases tab.'

Continuous deployment is not enabled by default when you create a new release definition from the **Releases** tab.

5. Open the **Tasks** tab, select the **Agent phase**, and choose **Remove** to remove this phase.

The screenshot shows the Pipeline Tasks tab. A red box highlights the "Agent phase" card, which has a "Run on agent" icon and a plus sign. To the right, a modal window titled "Agent phase" shows a "Display name" field containing "Agent phase". A red box highlights the "Remove" button at the top right of the modal.

6. Choose ... next to the **Environment 1** deployment process and select **Add deployment group phase**.

The screenshot shows the Pipeline Tasks tab. A red box highlights the "..." button next to the "Environment 1" deployment process. A dropdown menu appears with four options: "Add agent phase", "Add deployment group phase" (which is highlighted with a red box), "Add agentless phase", and "Learn more about phases".

7. For the **Deployment Group**, select the deployment group you created earlier such as **myNginx**.

The screenshot shows the Pipeline Tasks tab. A red box highlights the "Deployment group phase" card. In the configuration pane, a red box highlights the "Deployment group" dropdown menu, which shows "myNginx" selected. Other fields include "Display name" (set to "Deployment group phase") and "Deployment targets" (set to "Deployment targets ^").

The tasks you add to this phase will run on each of the machines in the deployment group you specified.

8. Choose + next to the **Deployment group phase** and, in the task catalog, search for and add a **Shell Script** task.

Pipeline    **Tasks** ▾    Variables    Retention    Options    History

Environment 1  
Deployment process

Deployment group phase **[+]**

Add tasks **shell script** **X**

- Azure CLI**  
Run a Shell or Batch script with Azure CLI commands against an azure subscription
- Azure PowerShell**  
Run a PowerShell script within an Azure environment
- PowerShell**  
Run a PowerShell script
- PowerShell on Target Machines**  
Execute PowerShell scripts on remote machine(s)
- Shell Script**  
Run a shell script using bash **Add**
- SSH**  
Run shell commands or a script on a remote machine using SSH

9. In the properties of the **Shell Script** task, use the **Browse** button for the **Script Path** to select the path to the **deploy.sh** script in the build artifact. For example, when you use the **nodejs-sample** repository to build your app, the location of the script is

```
$(System.DefaultWorkingDirectory)/nodejs-sample/drop/deploy/deploy.sh
```

Select File Or Folder

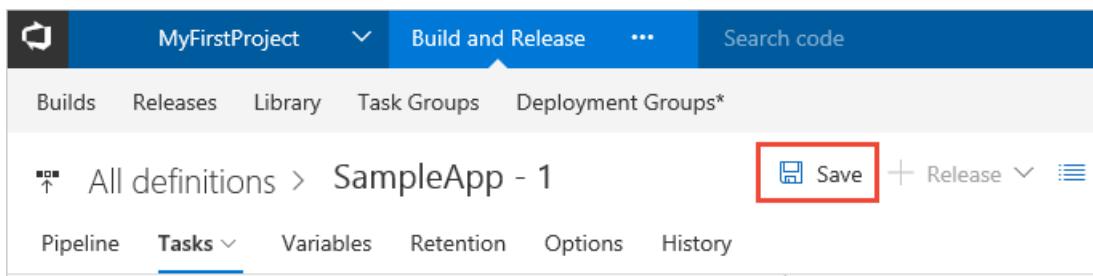
- Linked Artifacts
- myNodeJSApp (Build)
  - drop
    - deploy
      - deploy.sh
      - nginx-config
    - node\_modules
    - tests
    - Dockerfile
    - gulpfile.js
    - package.json
    - package-lock.json

The artifacts published by each version will be available for deployment in Release Management. The last successful version of myNodeJSApp (Build) published the following artifacts: *drop*.

Location **myNodeJSApp/drop/deploy/deploy.sh**

**OK** **Cancel**

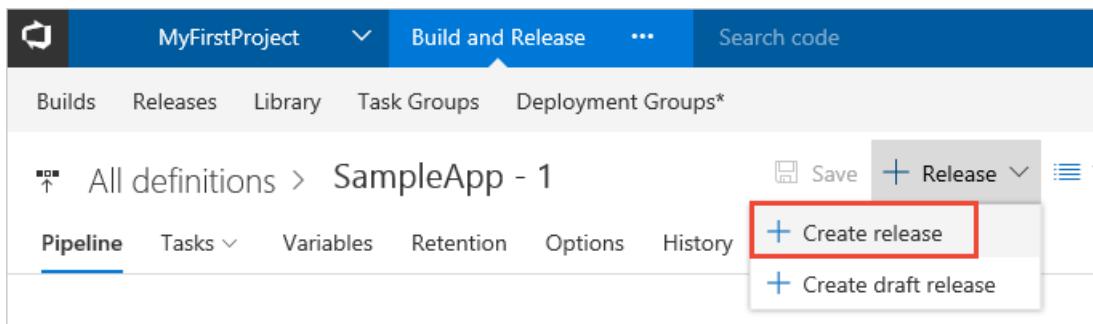
10. Save the release definition.



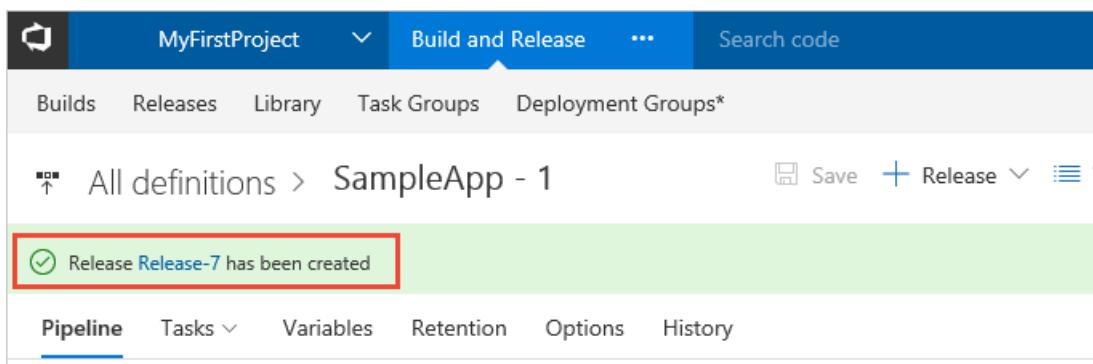
## Create a release to deploy your app

You're now ready to create a release, which means to start the process of running the release definition with the artifacts produced by a specific build. This will result in deploying the build.

1. To test the release definition, choose **Release** and then **Create release**.



2. In the Create new release panel, choose **Create**. Then choose the link near the top of the window that indicates a new release was created.



3. Open the **Logs** tab to watch the live logs from the deployment as it happens. Wait for the release to be deployed to the Azure web app.

SampleApp - 1 / Release-6

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

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

Step	Action
Environment 1	...
Pre-deployment approval	...
Deployment group phase	...
myVM	...
Initialize Job	...
Download Artifacts	...
Shell Script \$(System...	...
Post-deployment approval	...

DeploymentGroup: myNginx

Machines	Started Time	Finished Time	Duration	Status
myVM	11/17/2017...	11/17/2017...	00:02:58	Succeeded

4. After deployment is complete, open your web browser and test your web app using the URL

`http://<publicIpAddress>`, where `<publicIpAddress>` is the IP address of the web site on your nginx web server.

## Next steps

- [Dynamically create and remove a deployment group](#)
- [Apply environment-specific configurations](#)
- [Perform a safe rolling deployment](#)
- [Deploy a database with your app](#)

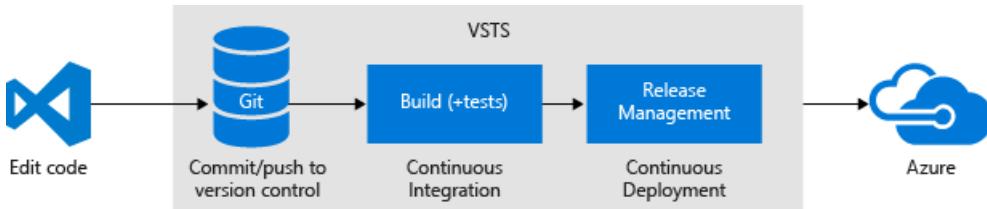
# Deploy to an Azure Web App for Containers

2/26/2018 • 4 min to read • [Edit Online](#)

## VSTS

We'll show you how to set up continuous deployment of your Docker-enabled app to an Azure web app using Visual Studio Team Services (VSTS).

For example, you can continuously deliver your app to a Windows VM hosted in Azure.



After you commit and push a code change, it is automatically built and then deployed. The results will automatically show up on your site.

## Define your CI build process

You'll need a continuous integration (CI) build process that publishes a Docker container image. To set up a CI build process, see:

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

## Prerequisites

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

## Create an Azure web app to host a container

1. Sign into your Azure Account at <https://portal.azure.com>.
2. In the Azure Portal, choose **Create a resource**, **New**, **Web + Mobile**, then choose **Web App for Containers**.
3. Enter a name for your new web app, and select or create a new Resource Group. Then choose **Configure container** and select **Azure Container Registry**. Use the drop-down lists to select the registry you created earlier, and the Docker image and tag that was generated by the build definition.

The screenshot shows the Azure portal interface for creating a Web App for Containers. The left pane displays the configuration for the app, including the app name (SampleApp), subscription (Visual Studio Enterprise), resource group (Create new, SampleApp), and app service plan/location (ServicePlan, (West...)). The right pane, titled 'Docker Container', shows the Docker configuration. It includes fields for 'Image source' (Azure Container Registry selected), 'Registry' (SampleApp1), 'Image' (dotnetcore-docker-sample), 'Tag' (70), and 'Startup File'. At the bottom, there is a checkbox for 'Pin to dashboard' and two buttons: 'Create' (highlighted in blue) and 'Automation options'.

The **Docker** tasks you used in the build definition when you created the build artifacts push the Docker image back into your Azure Container Registry. The web app you created here will host an instance of that image and expose it as a website.

4. Wait until the new web app has been created. Then you can create a release definition as shown in the next section.

#### Why use a separate release definition instead of the automatic deployment feature available in Web App for Containers?

You can configure Web App for Containers to automatically configure deployment as part of the CI/CD process so that the web app is automatically updated when a new image is pushed to the container registry (this feature uses a [webhook](#)). However, by using a separate release definition in VSTS or TFS you gain extra flexibility and traceability. You can:

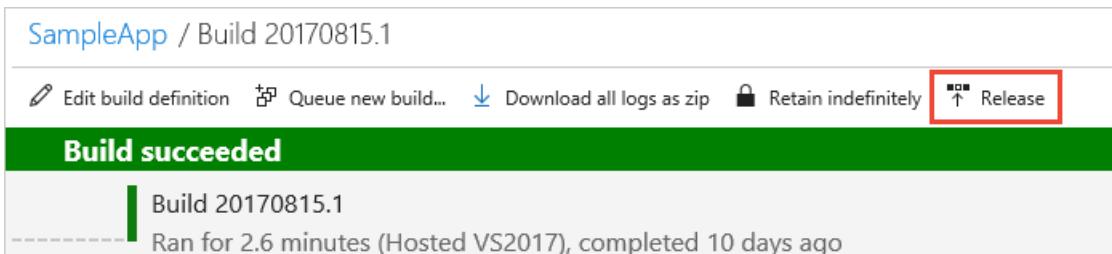
- Specify an appropriate tag that is used to select the deployment target for multi-environment deployments.
- Use separate container registries for different environments.
- Use parameterized start-up commands to, for example, set the values of variables based on the target environment.
- Avoid using the same tag for all the deployments. The default CD process for Web App for Containers uses the same tag for every deployment. While this may be appropriate for a tag such as **latest**, you can achieve end-to-end traceability from code to deployment by using a build-specific tag for each deployment. For example, the Docker build tasks let you tag your images with the **Build.ID** for each deployment.

## Create a release definition

1. In the **Build & Release** hub, open the build summary for your build.

Build Definitions			Build ID or build number
Mine	All Definitions	Queued	
Requested by me			Status
 MyFirstProject-ASP.NET Core-CI (1)	#20170907.1	<span style="border: 1px solid red; padding: 2px;">...</span>	<span style="color: green;">✓ succeeded</span>

2. In the build summary page, choose the **Release** icon to start a new release definition.



SampleApp / Build 20170815.1

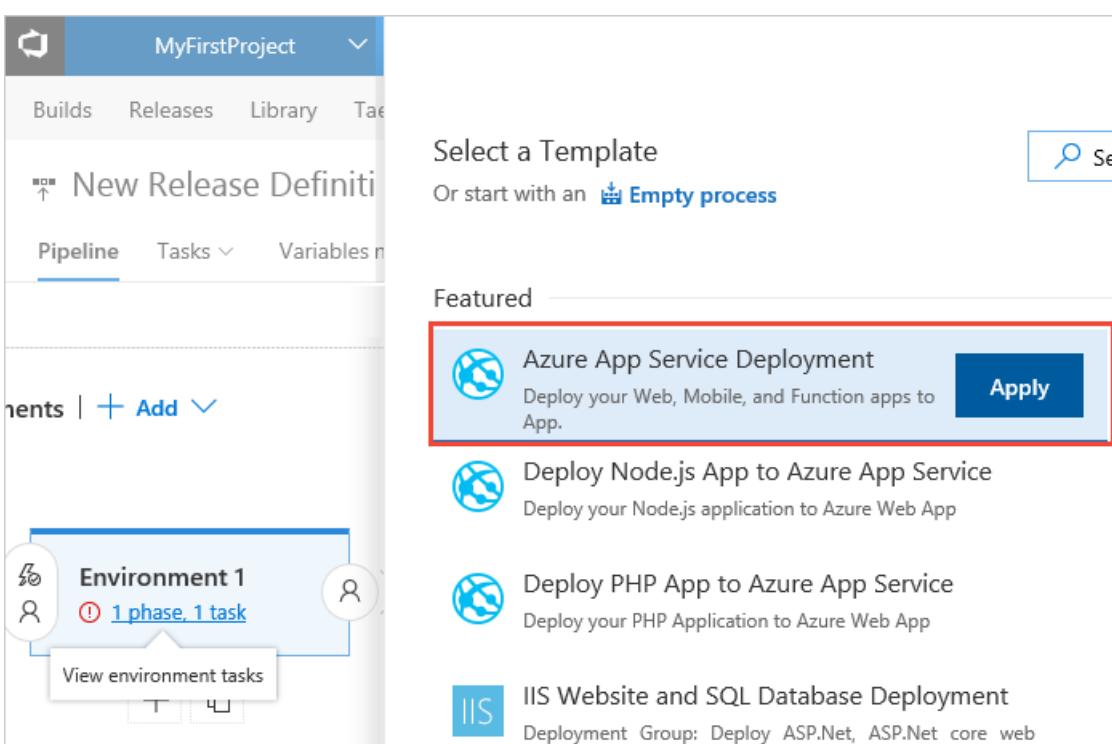
Edit build definition Queue new build... Download all logs as zip Retain indefinitely Release

**Build succeeded**

Build 20170815.1  
Ran for 2.6 minutes (Hosted VS2017), completed 10 days ago

If you have previously created a release definition that uses these build artifacts, you will be prompted to create a new release instead. In that case, go to the **Releases** tab page and start a new release definition from there by choosing the + icon.

3. Select the **Azure App Service Deployment** task and choose **Apply**.



New Release Definition

Builds Releases Library Tags

Pipeline Tasks Variables

Components | + Add

Select a Template

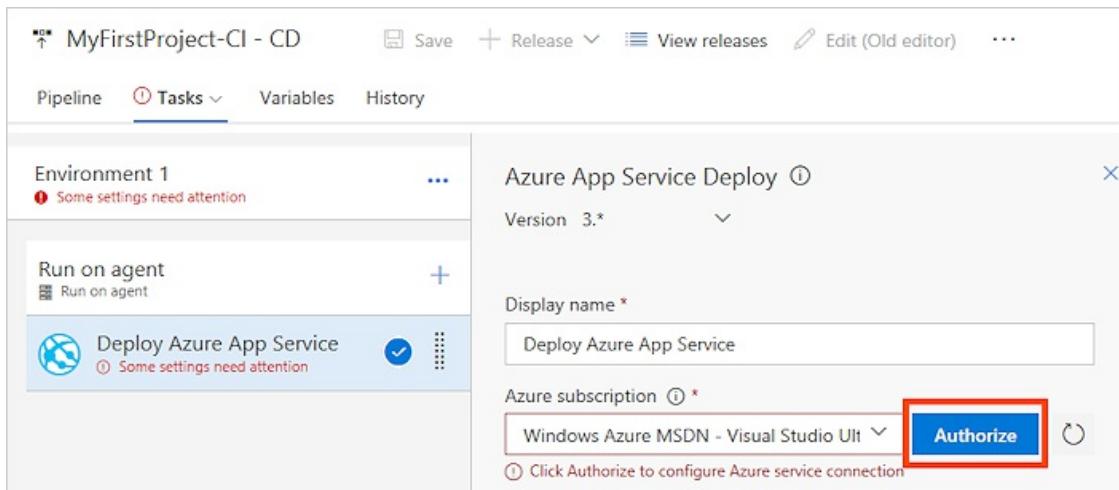
Or start with an Empty process

Featured

-  Azure App Service Deployment Deploy your Web, Mobile, and Function apps to App. Apply
-  Deploy Node.js App to Azure App Service Deploy your Node.js application to Azure Web App
-  Deploy PHP App to Azure App Service Deploy your PHP Application to Azure Web App
-  IIS Website and SQL Database Deployment Deployment Group: Deploy ASP.Net, ASP.Net core web

4. Open the **Tasks** tab and select the **Azure App Service Deployment** task. Configure the properties 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 VSTS and if you see an **Authorize** button next to the input, click on it to authorize VSTS 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 endpoint](#) to manually set up the connection.



- **App Service type:** Select **Web App for Containers**.
- **App Service Name:** Select the name of the web app from your subscription.

When you select the Docker-enabled app service, the task recognizes that it is a containerized app, and changes the property settings to show the following:

- **Registry or Namespace:** Enter the path to your Azure Container Registry. Typically this is `your-registry-name.azurecr.io`
- **Repository:** Enter the name of your repository, which is typically of the format  
`<account name>/<code-repo-name>`.
- **Tag:** Enter `$(Build.BuildId)`. The tag that was created automatically is the build identifier.

5. Save the release definition.

## 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 definition with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose **+ Release** and select **Create Release**.
2. In the **Create 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. Open the **Logs** tab to watch the release console output.
5. After the release is complete, navigate to your site running in Azure using the Web App URL  
`http://{web_app_name}.azurewebsites.net`, and verify its contents.

## Next steps

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

# Run tests with your builds

2/26/2018 • 1 min to read • [Edit Online](#)

**VS 2017 | VS 2015 | VSTS | TFS 2018 | TFS 2017 | TFS 2015**

Make sure that your app still works after every check-in and build using Visual Studio Team Services (VSTS) or Team Foundation Server (TFS). Find problems earlier by running tests automatically with each build. When your build is done, review your test results to start resolving the problems that you find.

This quickstart shows how to run unit tests with your build for .NET and ASP.NET apps. It uses the [Visual Studio Test task](#).

## Before you start

- [Check in your solution](#) to VSTS. Include your test projects.

## Create a build definition

Your build definition must include a test task that runs unit tests. For example, if you're building a Visual Studio solution in VSTS, your build definition should include a **Visual Studio Test** task. After your build starts, this task automatically runs all the unit tests in your solution - on the same build machine.

1. If your build definition does not contain a test task, add one to it.

The screenshot shows the 'Add tasks' interface in VSTS. On the left, there's a sidebar with a 'Process' section containing 'Build process' and a list of tasks: 'Get sources' (Fabrikam master), 'NuGet restore \*\*\\*.sln' (NuGet Installer), and 'Build solution \*\*\\*.sln' (Visual Studio Build). Below this is a button labeled '+ Add Task' with a red box around it. The main area is titled 'Add tasks' and has a sub-header 'Don't see what you need? Check out our Marketplace.' Below this is a navigation bar with tabs: All, Build, Utility, **Test**, Package, Deploy, Tool. Under the 'Test' tab, there are five listed tasks: 'Publish Code Coverage Results' (Publish Cobertura or JaCoCo code coverage results from a build), 'Publish Test Results' (Publish Test Results to VSTS/TFS), 'Run Functional Tests' (Run Coded UI/Selenium/Functional tests on a set of machines (using Test Agent)), 'Visual Studio Test' (Run tests with Visual Studio test runner), and 'Visual Studio Test Agent Deployment' (Deploy and configure Test Agent to run tests on a set of machines). The 'Visual Studio Test' task is also highlighted with a red box.

2. Edit the Visual Studio Test task. You can add filter criteria to run specific tests, enable code coverage, run tests from [other unit test frameworks](#), and so on.

The screenshot shows the Azure DevOps build pipeline editor. On the left, the 'Process' tab is selected, displaying the build steps: 'Get sources', 'NuGet restore', 'Build solution', 'VsTest - testAssemblies' (which is currently selected and highlighted with a dashed border), 'Publish symbols path', 'Copy Files to', 'Publish Artifact', and an 'Add Task' button. On the right, the 'Visual Studio Test' task is being configured. The task version is set to '2.\*'. The 'Display name' is 'VsTest - testAssemblies'. Under 'Test selection', 'Select tests using' is set to 'Test assemblies', with the pattern '\*\*\\$(BuildConfiguration)\\*test\*.dll !\*\*\obj\\*\*'. Other settings include 'Search folder' (\$System.DefaultWorkingDirectory) and 'Test filter criteria'. There are also two unchecked checkboxes: 'Run only impacted tests' and 'Test mix contains UI tests'.

For information about the option settings of the Visual Studio Test task, see:

- [Visual Studio Test version 1](#)
- [Visual Studio Test version 2](#)

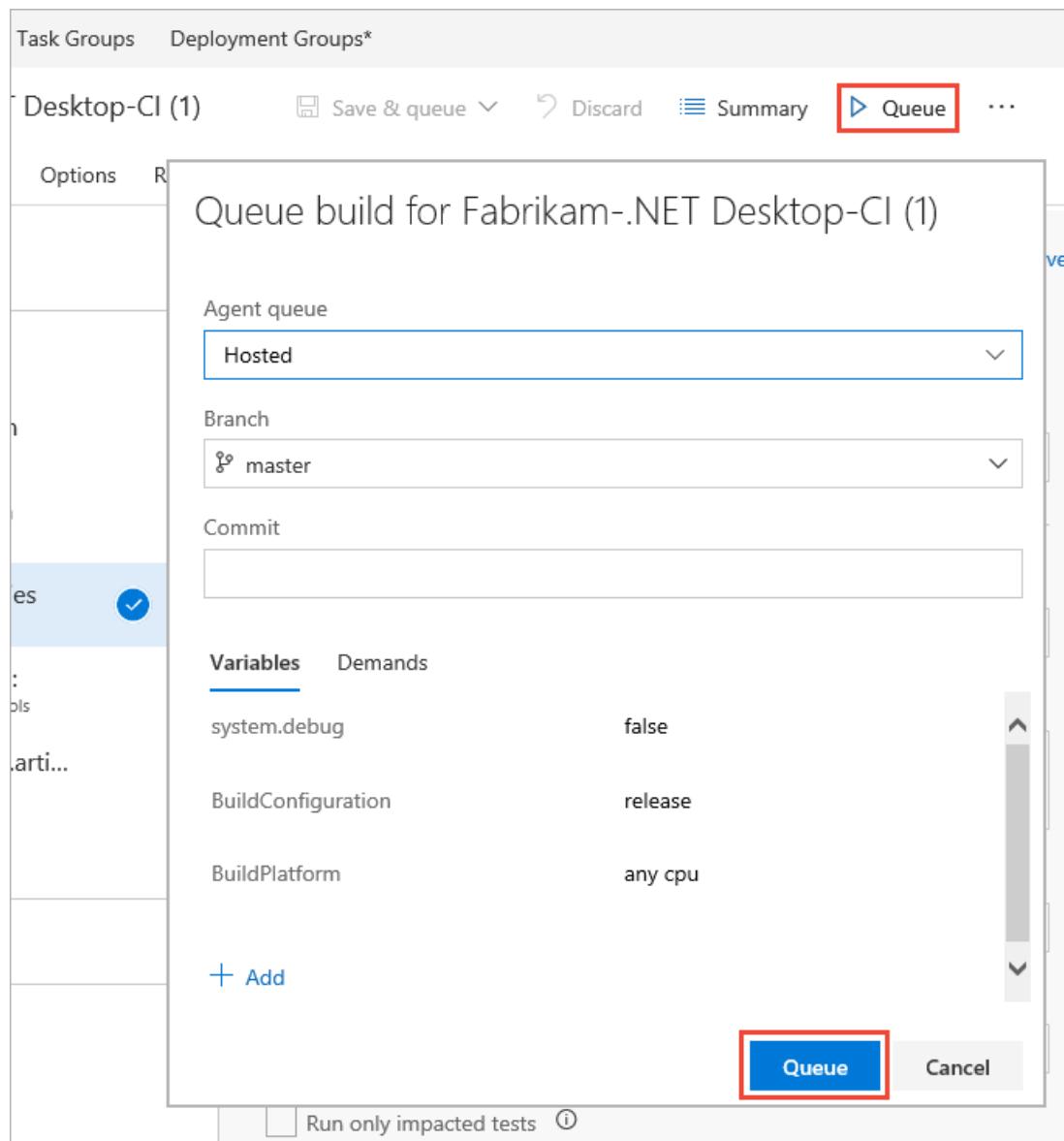
The Visual Studio Test task version 2 supports Test Impact Analysis (TIA). See [Speed up testing with Test Impact Analysis](#).

3. When you're done, save your build definition.

The screenshot shows the Azure DevOps build definition page for 'Fabrikam-.NET Desktop-CI (1)'. The 'Tasks' tab is selected. The build process consists of a single step: 'Get sources'. In the top right corner, there is a 'Save & queue' button. A dropdown menu is open from this button, showing three options: 'Save & queue' (which is the top item), 'Save' (which is highlighted with a red box), and 'Save as draft'.

## Start the build

1. Start the build by adding it to the build queue.



2. After the build finishes, you can review the test results to resolve any problems that happened. Go to the build to open the build summary.

The screenshot shows the 'Build Definitions' summary page for 'Fabrikam-.NET Desktop-CI (1)'. The 'Summary' tab is selected. The 'Details' section shows the repository is 'Fabrikam' (Hosted | Manage), last updated on Friday, June 30, 2017, at 10:51 AM. The 'Queued & running' section indicates 'No builds queued or running at the moment'. The 'Recently completed' section shows a single build: '#20170630.1' (succeeded, master branch). The 'Analytics' section is partially visible at the bottom.

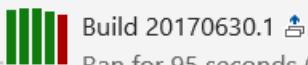
## Review the results

1. Open the test run results summary and compare your test results between this build and the last build. Here you'll find changes in new, failed, and passed tests, how long these tests took to run, how long these tests have been failing, and more. Organize your test results and open bugs directly for failed tests.

## Fabrikam-.NET Desktop-CI (1) / Build 20170630.2 / Build

[Edit build definition](#) [Queue new build...](#) [Download all logs as zip](#) [Release](#)

### Build failed



Build 20170630.1 [View details](#)

Ran for 95 seconds (Hosted), completed 12 seconds ago

[Summary](#) [Timeline](#) [Artifacts](#) [Code coverage\\*](#) [Tests](#)

Total tests	Failed tests	Pass percentage	Run duration
3 (+0) ■ Passed (2) ■ Failed (1) ■ Others (0)	1 (+1) ■ New (1) ■ Existing (0)	66.66% (-33.4%)	753ms (-243ms)

[New](#) [Edit](#) [Delete](#) [Copy](#) [Link](#)

Group by [Test run](#)

Test	Failing since	Failing build	Duration
2/3 Passed - VSTest Test Run release any cpu / About	just now	Current build	0:00:00.123
About <a href="#">New</a>	just now	Current build	0:00:00.000
Contact			0:00:00.000
Index			0:00:00.273

Outcome All [Filter](#)

- Failed
- Passed
- Not Impacted
- Others
- All

- To start debugging a failed test, open the test and review the resulting error and stack trace.

## Run 52 - VSTest Test Run release any cpu / About

[Summary](#) [History](#)

[Create bug](#) | [Update analysis](#)

### Summary

[Failed](#) on FACTORYVM-41

Run by [\[REDACTED\]](#)  
Tested build [20170630.2](#)  
Priority not available  
Configuration not available

### Analysis

Owner not available  
Failure type None  
Resolution None  
Comment not available

### Error message

```
Assert.AreEqual failed. Expected:<My wonderful new web app.>. Actual:<My super new web app.>.
```

### Stack trace

```
at CallSite.Target(Closure , CallSite , Type , String , Object )
at System.Dynamic.UpdateDelegates.UpdateAndExecuteVoid3[T0,T1,T2](CallSite site, T0 arg0, T1 arg1, T2 arg2)
at SampleWebAppWithTests.Tests.Controllers.HomeControllerTest.About() in d:\a\1\s\SampleWebAppWi...
```

### Attachments (0)

No attachments

### Bugs (0)

No linked bugs

### Requirements (0)

No linked requirements

## Next step

[Review your test results](#)

# Define a continuous integration (CI) build process for your Git repo

2/28/2018 • 6 min to read • [Edit Online](#)

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a full-featured Git server for hosting your team's source code. To keep code quality high, add continuous integration (CI) builds to your team's process. CI builds automatically build and test code every time a team member pushes a commit to the server. You can take it a step further with pull request builds.

In this tutorial, you learn how to:

- Set up a CI trigger for feature branches
- Execute CI for a topic branch
- Exclude or include tasks for builds based on the branch being built
- Keep code quality high by building your pull requests
- Use retention policies to clean up your completed builds

## Prerequisites

- Git repository in VSTS or TFS
- A working build definition for a Git repository in VSTS
  - If needed, complete one of the following: [Build and deploy to an Azure Web App](#), [Build your Java app with Maven](#), or [Build your Node.js with Gulp](#).

## Set up a CI trigger for a topic branch

A common workflow with Git is to create temporary branches from your master branch. These branches are called topic or feature branches and help you isolate your work. In this workflow, you create a branch for a particular feature or bug fix. Eventually, you merge the code back to the master branch and delete the topic branch. VSTS allows you to create and delete topic branches to take advantage of CI without having to edit the build definition. You can use naming conventions, wildcards, and branch filters to initiate builds that match a particular pattern. Follow the steps below to create a CI trigger that will execute a build for feature branches.

1. Select **Build and Release**, and then choose **Builds**.
2. Locate the **Build Definition** that services your master branch. Select the **ellipsis** to the right of your definition. Select **Edit**.
3. Select the **Triggers** menu for your build. Ensure you have **Continuous Integration** enabled.
4. Select the **+ Add** icon under **Branch filters**.
5. Under the **Branch specification** dropdown, type **features/\*** in the **Filter my branches** text box and press **Enter**. The trigger now supports CI for all feature branches that match the wildcard as well as the master

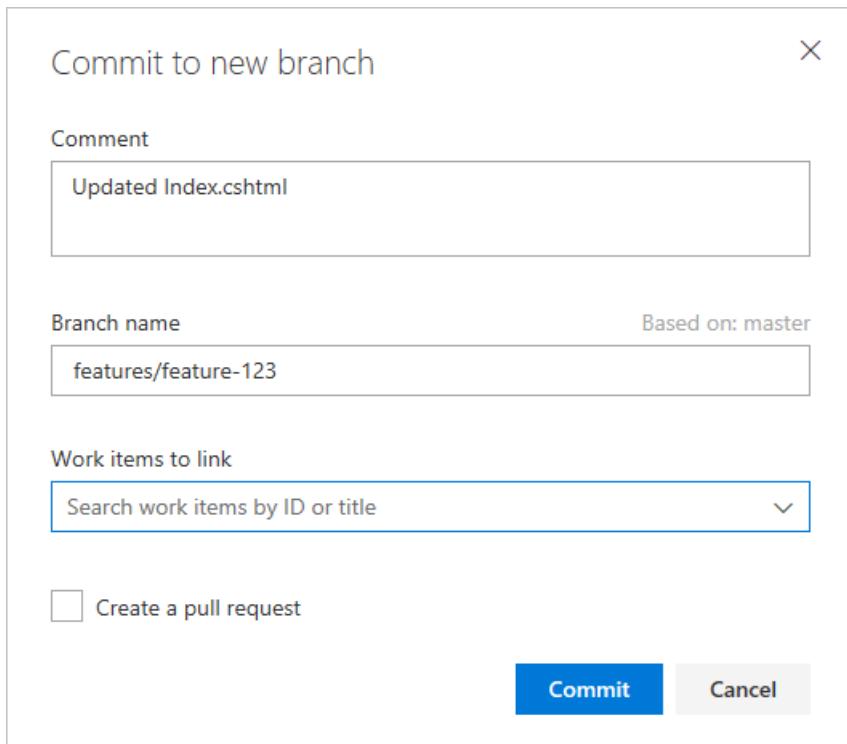
The screenshot shows the 'Triggers' tab of a VSTS build definition named 'DevOpsDemos-ASP.NET Core (.NET Framework)-CI'. Under the 'Continuous Integration' section, it is configured to 'Build every change to matching branches'. The 'Enabled' toggle switch is turned on. The 'Repositories' section lists a single repository named 'dotnetcore-sample'. For this repository, the 'Build when any branch changes' option is selected, and the 'Batch changes while a build is in progress' checkbox is unchecked. The 'Branch Filters' section contains two entries: 'master' and 'features/\*', both set to 'Include' type. There are also 'Path Filters' and 'Add' buttons for further configuration. A note at the bottom left says 'branch.'

6. Select the **Save & queue** menu and then Select **Save**.

## Execute CI for a topic branch

Your build definition is now ready for CI for both the master branch and future feature branches that match the branch pattern. Every code change for the branch will use an automated build process 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 the **Code** hub in VSTS.
2. Choose your **repository** and Select **Branches**. Choose the **master branch**.
3. Select the **Files** menu. Make a quick code change by selecting a file and Selecting **Edit**. Add some text and Select **Commit**. For **Branch name**, remove master and type **features/feature-123**.
4. Select **Commit**. This workflow creates a new topic branch under a parent named **features** and commits your code edits to the new branch.



5. Navigate to the **Build and Release** menu in VSTS and Select **Builds**.
6. Select **Queued** under **Build Definitions** to view the queued builds. You should now see your new build definition 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 typically includes developing code locally and periodically pushing to your remote topic branch. Each push you make will result in a build process executing in the background. The build process helps you catch errors earlier and helps you to maintain a quality topic branch that can be safely merged to master. Practicing CI for your topic branches helps to minimize risk when merging back to master.

## Exclude or include tasks for builds based on the branch being built

The master branch typically produces deployable artifacts such as binaries. You do not need to spend time creating and storing those artifacts for short-lived feature branches. You implement custom conditions in VSTS so that certain tasks only execute on your master branch during a build run. You can use a single build with multiple branches and skip or perform certain tasks based on conditions.

1. Select **Build and Release** menu and Select **Builds**.
2. Locate the **Build Definition** that services your master branch. Select the **ellipsis** to the right of your definition. Select **Edit**.
3. Choose the **Publish Artifact** task in your build definition.
4. Select **Control Options** on the bottom right hand part of your screen.
5. Select the dropdown for **Run this task** and choose **Custom conditions**.

Publish Build Artifacts

Version 1.\*

Display name \* Publish Artifact

Path to Publish \* \$(build.artifactstagingdirectory)

Artifact Name \* drop

Artifact Type \* Server

Control Options

- Enabled
- Continue on error

Timeout \* 0

Run this task

Custom conditions

Custom condition

```
and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
```

- Enter the following snippet: `and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))`
- Select the **Save & queue** menu, and then select **Save & queue**.
- Choose your **topic branch**. Select **Queue**. We are not building the master branch, and the task for **Publish artifacts** will not execute.
- Select the build to monitor the progress. Once the build completes, confirm the build skipped the **Publish artifacts** task step.

## Keep code quality high by building your pull requests

Use policies to protect your branches by requiring successful builds before merging pull requests. You have options to always require a new successful build before merging changes to important branches such as the master branch. There are other branch policy settings to build less frequently. You can also require a certain number of code reviewers to help ensure your pull requests are high quality and don't result in broken builds for your branches.

1. Navigate to the **Code** hub in VSTS.
2. Choose your **repository** and Select **Branches**. Choose the **master branch**.
3. You will implement a branch policy to protect the master branch. Select the **ellipsis** to the right of your branch name and Select **Branch policies**.
4. Choose the checkbox for **Protect this branch**. There are several options for protecting the branch.
5. Under the **Build validation** menu choose **Add build policy**.
6. Choose the appropriate **build definition**.
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 VSTS.
11. Select **New pull request**. Ensure your topic branch is set to merge into your master branch. Select **create**.
12. Your screen displays the **policy** being executed.

13. Select the **policy name** to examine the build. If the build succeeds your code will be merged to master. If the build fails the merge is blocked.

Once the work is completed in the topic branch and merged to master, you can delete your topic branch. You can then create additional feature or bug fix branches as necessary.

## Use retention policies to clean up your completed builds

Retention policies allow you to control and automate the cleanup of your various builds. For shorter-lived branches like topic branches, you may want to retain less history to reduce clutter and storage costs. If you create CI builds on multiple related branches, it will become less important to keep builds for all of your branches.

1. Navigate to the **Build and Release** menu in VSTS.
2. Select the **Build** that you set up for your topic branch.
3. Select **Edit** at the top right of your screen.
4. Under the build definition name, Select the **Retention** tab. Select **Add** to add a new retention policy.

The screenshot shows the 'Retention' tab in the VSTS build definition editor. On the left, a list of existing policies is shown:

- 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)

On the right, the 'Settings' section is configured as follows:

- Branch Filters:** Type is set to 'Include' and the 'Branch specification' dropdown contains 'features/\*'.
- Days to keep:** Set to 1.
- Minimum to keep:** Set to 1.
- When cleaning up builds, delete the following:** All checkboxes are selected: Build record, Source label, File Share, Symbols, and Automated test results.

5. Type **features/\*** in the **Branch specification** dropdown. This ensures any feature branches matching the wildcard will use the policy.
6. Set **Days to keep** to 1 and **Minimum to keep** to 1.
7. Select the **Save & queue** menu and then Select **Save**.

Policies are evaluated in order, applying the first matching policy to each build. The default rule at the bottom matches all builds. The retention policy will clean up build resources each day. You retain at least one build at all times. You can also choose to keep any particular build for an indefinite amount of time.

## Next steps

In this tutorial, you learned how to set up and manage CI with Git and VSTS.

You learned how to:

- Set up and execute CI for a feature branch
- Exclude or include tasks for builds based on the branch being built
- Keep code quality high by building your pull requests
- Use retention policies to clean up your completed build

[Deploy to Azure web app](#)

# Define a continuous integration (CI) build process for your GitHub repository

2/13/2018 • 6 min to read • [Edit Online](#)

Visual Studio Team Services (VSTS) can perform continuous integration (CI) and continuous delivery (CD) for code in your GitHub repository.

In this tutorial, you learn how to:

- Set up CI builds for your GitHub repository
- Create a VSTS build status with a GitHub README file
- Create a pull request trigger for GitHub

## Prerequisites

- A VSTS account. 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 team project you want to use.
- Ensure you have GitHub contributor rights for a repository.
- Fork this [sample repository](#) into your GitHub account

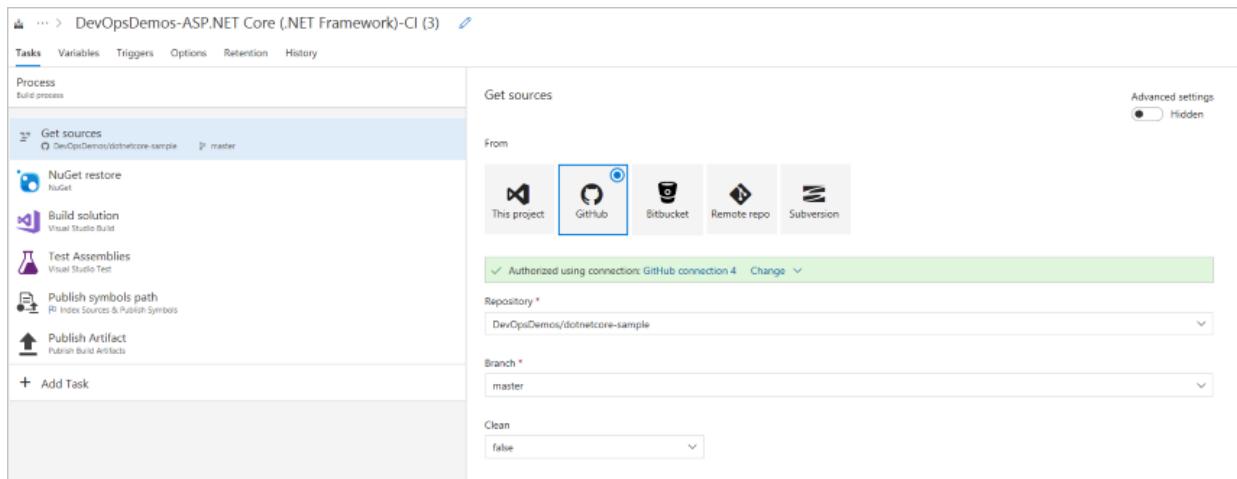
## Set up CI builds for your GitHub repository

You can manage your application code in GitHub, and configure continuous integration using VSTS. Build every commit in GitHub by adding the trigger and enabling CI for the build definition. Follow the steps below to configure GitHub as a source for your VSTS build.

### IMPORTANT

Ensure your browser does not block the pop-up on step 7 below.

1. Navigate to your VSTS account and team project. Select **Build and Release**, and then select **Builds**.
2. Select **New** to create a new build definition.
3. Select the **ASP.NET Core (.NET Framework)** build template.
4. Choose **Hosted VS2017** for Agent queue.
5. Choose the **Get sources** task for your build definition.
6. Choose **GitHub** from **Get sources**.



7. Give your connection a name, and then select the **Authorize using OAuth** button. Optionally you can use a GitHub **personal access token** instead of OAuth.
8. When prompted, sign in to your **GitHub account**. Then select **Authorize** to grant access to your VSTS account.
9. Choose the **repository** that contains the sample you forked earlier.
10. Select **Triggers**. Under **Continuous integration**, select on the name of your repository.
11. Toggle on the checkbox labeled **Enable continuous integration**. Ensure you include the **master branch** under **Branch filters**. This setting ensures each commit to `master` in GitHub will trigger a build via a **GitHub webhook**.
12. **Note:** The trigger setting for CI is for the purpose of this tutorial, and you can choose to leave this setting off to have fewer builds.
13. Select **Options**, and then toggle on the checkbox to enable the **Badge enabled** setting.
14. Select **Save & queue**, then select **Save** to save your build definition.
15. Copy the URL under the **Badge enabled** setting. You will use this URL in upcoming steps. The URL displays a build status badge that can be safely viewed in a browser or used in external applications to display the build status of your VSTS build definition.

## Create a VSTS build status with a GitHub README file

This section explores possibilities for further integrating VSTS and GitHub. You will create a notification for the VSTS build definition by populating a GitHub readme file with Markdown that points to the build notification URL. This same Markdown can link back to the build summary page in VSTS.

1. Navigate to your GitHub account. Select **Code**. Create a README.md file unless one already exists.
2. For this step, use the URL from step 13 in the previous section. Modify the **README.md** file by adding Markdown and an `img` tag, as shown below. Create a Markdown link that displays the build status image and links to the build summary in VSTS. **Modify** the below example with your account and build ID information.

```
[]
(https://{{your-account}}.visualstudio.com/{{your-project}}/_build/index?definitionId={{id}})
```

3. **Commit** your README.md file to the repository. The code tab will now display the current status of your VSTS build.

The screenshot shows a GitHub repository page. At the top, there are navigation links: Code, Pull requests (1), Projects (0), Wiki, Insights, and Settings. Below the header, it says "No description, website, or topics provided." with an "Edit" button. There's a link to "Add topics". Below this, there are summary statistics: 14 commits, 2 branches, 0 releases, and 3 contributors. A green progress bar indicates the repository is 5 commits ahead of the master branch. Below the stats, there are buttons for "Branch: master", "New pull request", "Create new file", "Upload files", "Find file", and "Clone or download". A note says "This branch is 5 commits ahead, 2 commits behind adventworks:master." with links to "Pull request" and "Compare". The commit list starts with a commit from "mlhoop" titled "fix breaking change" (latest commit 5f08f0f, 21 days ago). Other commits include "dotnetcore-sample" (fix breaking change, 21 days ago), "dotnetcore-tests" (Add files via upload, 2 months ago), ".gitattributes" (First commit, 4 months ago), ".gitignore" (First commit, 4 months ago), "README.md" (added build status verbiage, 21 days ago), and "dotnetcore-sample.sln" (Add test project, 3 months ago). Below the commit list is a section for "README.md".

## Build Status from VSTS

build succeeded

4. In GitHub, navigate to **Settings** and select on **Webhooks**. You should see the webhook that was created by VSTS when you chose GitHub in your build definition. Every commit to the GitHub repository will trigger a build in VSTS.
5. Navigate back to your **VSTS build definition**. Under **Build and Release** choose **Builds**. You should notice a build is in progress or has completed for your build definition because of your `README.md` commit in the GitHub repository.
6. View the build summary, and then select the **Commit ID** link under the **Associated changes** section. This link navigates you directly to the GitHub commit.

The screenshot shows a VSTS build summary page for a build named "MSFTDemos-ASP.NET Core (.NET Framework)-CI (edit)". The build was triggered by a GitHub commit (5f08f0f) and completed successfully on Wednesday, September 20, 2017, at 5:47 PM. The build summary includes sections for "Build details", "Associated changes", "Work items linked to associated changes", "Test Results", "Code Coverage", "Tags", and "Deployments".

Build details	Associated changes	Test Results	Code Coverage	Tags	Deployments
Definition: MSFTDemos-ASP.NET Core (.NET Framework)-CI (edit)	5f08f0f Authored by mlhoop fix breaking change	No test runs are available for this build. Enable automated tests in your build definition by adding the <a href="#">Visual Studio Test task</a> .	Code coverage data is shown only for completed builds	Add tag...	No deployments found for this build. <a href="#">Create release</a> .
Source: master					
Source version: 5f08f0f					
Requested by: Microsoft.VisualStudio.Services.TFS on behalf of Micheal Learned					
Queue name: Hosted VS2017					
Queued: Wednesday, September 20, 2017 5:47 PM					
Started: Wednesday, September 20, 2017 5:47 PM					
Finished: Build not retained					
Retained state: Build not retained					

## Create a pull request trigger for GitHub

In this section, you create a pull request trigger for your VSTS build definition. Enabling the trigger in VSTS allows you to initiate CI builds for every GitHub pull request. Even if you're using continuous integration (CI) on your development branches to catch problems early, building pull requests helps reduce issues before they arrive in your development branches.

1. Navigate to your VSTS account and team project. Select **Build and Release**, and then select **Builds**. Locate

- your build, and select **Edit**.
2. Select **Triggers**. Enable the **Pull request validation** trigger. Ensure you include the **master branch** under **Branch filters**.
  3. Select **Save & queue**, then select **Save**.
  4. Navigate to your GitHub account. Navigate to the main page for your **repository**.
  5. Select the **Branch** selector, and then type a name for a new branch and press enter. This will create a branch based on master.
  6. Edit a file in your new branch. **Commit** your change to the new branch.
  7. Select **Pull requests**. Select **New pull request**.
  8. Create the pull request. Navigate back to your **VSTS build definition**. A build will be queued or completed for the merge commit of your pull request.

## Building pull requests from repository forks

### IMPORTANT

These settings impact the security of your build.

To have VSTS automatically build pull requests from forks of your repository, enable the checkbox labeled **Build pull requests from forks of this repository**.

### Security considerations

By default, secrets associated with your build definition are not made available to builds of pull requests from forks. Secrets include:

- A security token with access to your GitHub repository
- These items, if used by your build:
  - [Service endpoint](#) credentials
  - Files from the [Secure Files library](#)
  - Build [variables](#) marked **secret**

If your build definition is configured with secrets, these builds will immediately fail. To bypass this precaution, enable the checkbox labeled **Make secrets available to builds of forks**. Be aware of this setting's impact on security as described below.

A GitHub user can fork your repository, change it, and create a pull request to propose changes to your repository. Such a pull request could contain malicious code to run as part of your triggered build. For example, an ill-intentioned script or unit test change could leak secrets or compromise the agent machine performing the build. The following steps are recommended to mitigate this risk:

1. Use a VSTS [hosted agent](#) to build pull requests from forks. Hosted agents are immediately deleted after they complete a build, so there is no lasting impact if they are compromised.
2. If you must use a [private agent](#), do not store secrets or perform other builds or releases on the same agent, unless your repository is private and you trust pull request creators. Otherwise, secrets could leak and the repository contents or secrets of other builds and releases could be revealed.
3. If your repository is public, do not enable the checkbox labeled **Make secrets available to builds of forks**. Otherwise, secrets could leak during a build.

## Q&A

[How do I use a personal access token to authorize the VSTS to GitHub connection?](#)

See this [article](#) for creating a GitHub personal access token. You can use the token in the VSTS **Get sources** task of your build or release definitions by creating a GitHub [service endpoint](#) and entering the token.

## Next steps

In this tutorial, you learned how to set up and manage CI with GitHub and VSTS.

You learned how to:

- Set up CI builds for your GitHub repository
- Display a VSTS build status within a GitHub README file
- Create a pull request trigger for GitHub

[Deploy to Azure Web App](#)

# Define your multi-stage continuous deployment (CD) process

2/28/2018 • 9 min to read • [Edit Online](#)

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) provide a highly configurable and manageable pipeline for releases to multiple environments such as development, staging, QA, and production environments; including requiring approvals at specific stages.

In this tutorial, you learn about:

- Configuring triggers within the release pipeline
- Extending a release definition by adding environments
- Configuring the environments as a multi-stage release pipeline
- Adding approvals to your release pipeline
- Creating a release and monitoring the deployment to each environment

## Prerequisites

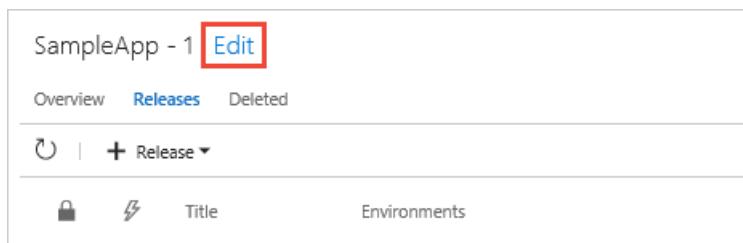
You'll need:

- A release definition that contains at least one environment. 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](#)
  - [Build and Deploy to an Azure Web App](#)
  - [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. If you need help, follow the steps in [the ASP.NET Core web app deployment tutorial](#).

## Configure the triggers in your release definition

In this section, you will check that the triggers you need for continuous deployment are configured in your release definition.

1. In the **Build & Release** hub, open the **Releases** tab. Select your release definition and, in the right pane, choose **Edit**.



2. Choose the **Continuous deployment trigger** icon in the **Artifacts** section to open the trigger panel. Make sure this is enabled so that a new release is created after every new successful build is completed.

The screenshot shows the Azure DevOps Pipeline tab for a release definition named "SampleApp - 1". On the left, there are sections for "Artifacts" and "Environments". In the "Environments" section, "Environment 1" is listed with the description "1 phase, 1 task". On the right, the "Continuous deployment trigger" settings are displayed. A red box highlights the "Enabled" toggle switch, which is turned on. Below it, a tooltip says "Creates release every time a new build is available." Other settings include "Build branch filters" and "Build tags".

For more information, see [Release triggers](#) in the Release Management documentation.

- Choose the **Pre-deployment conditions** icon in the **Environments** section to open the conditions panel. Make sure that the trigger for deployment to this environment is set to **Release**. This means that a deployment will be initiated automatically when a new release is created from this release definition.

The screenshot shows the Azure DevOps Pipeline tab for the same release definition. In the "Environments" section, "Environment 1" is selected. On the right, the "Pre-deployment conditions" settings are shown for the "Production" environment. A red box highlights the "Release" trigger option under "Select the source of the trigger". Other options include "Manual only", "Artifact filters", and "Schedule".

Notice that you can also define artifact filters that determine a condition for the release to proceed, and set up a schedule for deployments. You can use can features to, for example, specify a branch from which the build artifacts must have been created, or a specific time of day when you know the app will not be heavily used. For more information, see [Environment triggers](#) in the Release Management documentation.

## Extend a release definition by adding environments

In this section, you will add a new environment to the release definition. The two environments will deploy your app to the "QA" and the "Production" targets (in our example, two Azure App Services websites). This is a typical scenario where you deploy initially to a test or staging server, and then to a live or production server. Each [environment](#) represents one deployment target, though that target could be a physical or virtual server, a groups of servers, or any other legitimate physical or virtual deployment target.

- In the **Pipeline** tab of your release definition, select the existing environment and rename it to

## Production.

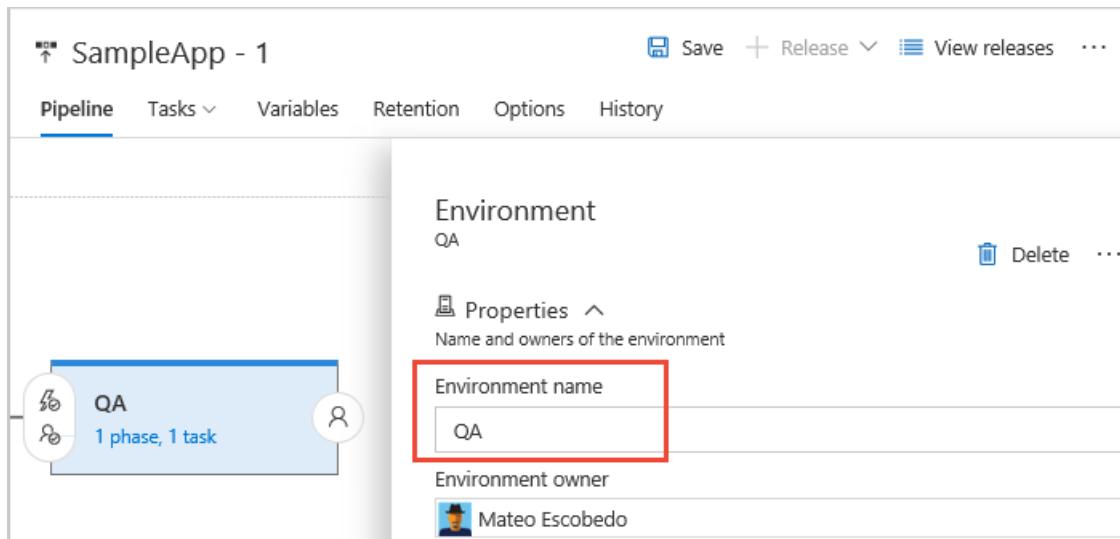
The screenshot shows the 'SampleApp - 1' pipeline configuration. On the left, there's a 'Pipeline' tab with a 'Build' and 'Drop' artifact section. On the right, under 'Environments', the 'Production' environment is selected. The 'Properties' panel shows the environment name is 'Production'. The 'Environment owner' is listed as Mateo Escobedo. A red box highlights the 'Production' text in the 'Environment name' field.

2. Open the **+ Add** drop-down list and choose **Clone environment** (the clone option is available only when an existing environment is selected).

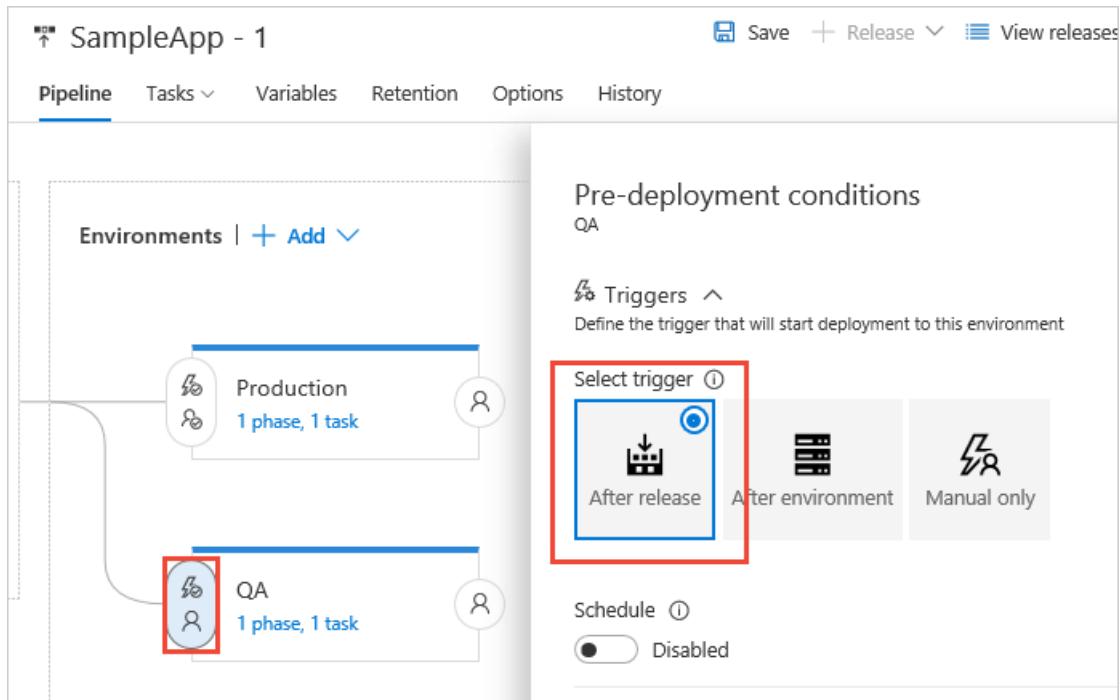
The screenshot shows the 'SampleApp - 1' pipeline configuration. The 'Pipeline' tab is selected. In the 'Environments' section, a dropdown menu is open over the 'Production' environment card. The menu has two options: '+ New environment' and '**Clone environment**'. The 'Clone environment' option is highlighted with a red box.

Typically, you want to use the same deployment methods with a test and a production environment so that you can be sure the deployed apps will behave in exactly the same way. Therefore, cloning an existing environment is a good way to ensure you have the same settings for both. Then you just need to change the deployment targets (the websites where each copy of the app will be deployed).

3. The clone of the environment appears after the existing environment in the pipeline, and has the name **Copy of Production**. Select this environment and, in the **Environment** panel, change the name to **QA**.



4. To reorganize the environments in the pipeline, choose the **Pre-deployment conditions** icon for the **QA** environment and set the trigger to **After release**. The pipeline diagram changes to show that the deployment to the two environments will now execute in parallel.



5. Choose the **Pre-deployment conditions** icon for the **Production** environment and set the trigger to **After environment**, then select **QA** in the **Environments** drop-down list. The pipeline diagram changes to show that the deployment to the two environments will now execute in the required order.

**Environments | + Add ▾**

QA  
1 phase, 1 task

Production  
1 phase, 1 task

**Pre-deployment conditions**  
Production

**Triggers**  
Define the trigger that will start deployment to this enviro

Select trigger ⓘ

- After release
- After environment**
- Manual or

Environments ⓘ

- QA**

Trigger even when selected environments pa

Notice that you can specify deployment to start when a deployment to the previous environment is *partially* successful. Usually, this means the deployment tasks were set to continue the deployment even if a specific non-critical task failed (the default is that all tasks must succeed). You're most likely to set this option if you create a pipeline containing [fork and join deployments](#) that deploy to different environments in parallel.

6. Open the **Tasks** drop-down list and choose the **QA** environment.

**SampleApp - 1**

**Pipeline** **Tasks** ⓘ Variables Retention Options History

**Artifacts** | + Add

**environments | + Add ▾**

Build Drop

QA

Production

QA  
1 phase, 1 task

Production  
1 phase, 1 task

7. Recall that this environment is a clone of the original **Production** environment in the release definition. Therefore, currently, it will deploy the app to the same target as the **Production** environment. Depending on the tasks that you are using, change the settings so that this environment deploys to your "QA" target. In our example, using Azure App Services websites, we just need to select the **Deploy Azure App Service** task and select the "QA" website instead of the "Production" website.

The screenshot shows the 'Tasks' tab of a release definition named 'SampleApp - 1'. It includes sections for 'QA Deployment process' and 'Phase1 Run on agent'. The 'Deploy Azure App Service' task is highlighted with a red box. The 'App Service name' dropdown is also highlighted with a red box, showing 'SampleAppQA' as the selected option.

If you are using a different type of task to deploy your app, the way you change the target for the deployment may differ. For example, if you are using deployment groups, you may be able to select a different deployment group, or a different set of tags within the same deployment group.

## Add approvals within a release definition

The release definition you have modified deploys to test and then to production. If the deployment to test fails, the trigger on the production environment does not fire, and so it is not deployed to production. However, it is typically the case that you want the deployment to pause after *successful* deployment to the test website so that you can verify the app is working correctly before you deploy to production. In this section, you will add an approval step to the release definition to achieve this.

- Back in the **Pipeline** tab of the release definition, choose the **Pre-deployment conditions** icon in the **Environments** section to open the conditions panel. Scroll down to the **Pre-deployment approvers** section and enable pre-deployment approvals.

The screenshot shows the 'Pipeline' tab of the 'SampleApp - 1' release definition. It displays environments: 'QA' (selected), 'Production' (disabled), and 'Manual only'. The 'Pre-deployment approvers' section is highlighted with a red box, showing the toggle switch is 'Disabled'.

- In the **Approvers** section, choose your account from the list. You can type part of a name to search for

matches. Also make sure you clear (untick) the checkbox **User requesting a release...** so that you can approve your own releases.

The screenshot shows the 'Pipeline' tab selected in the 'SampleApp - 1' release definition. In the 'Triggers' section, the 'Pre-deployment approvers' step is enabled. A user named 'mated' has been selected as an approver. Approval policies are set to prevent the approver from approving their own releases. A note indicates that these changes apply to post-deployment approvals as well.

You can add as many approvers as you need, both individual accounts and account groups. It's also possible to set up post-deployment approvals by choosing the icon at the right side of the environment item in the pipeline diagram. For more information, see [Approvals and gates overview](#).

3. Save the modified release definition.

The screenshot shows the 'Tasks' tab selected in the 'SampleApp - 1' release definition. The 'Save' button is highlighted with a red box.

## Create a release

Now that you have completed the modifications to the release definition, it's time to start the deployment. To do this, you create a release from the release definition. A release may be created automatically; for example, the continuous deployment trigger is set in the release definition. This means that modifying the source code will start a new build and, from that, a new release. However, in this section you will create a new release manually.

1. Open the **Release** drop-down list and choose **Create release**.

The screenshot shows the 'Tasks' tab selected in the 'SampleApp - 1' release definition. The 'Release' dropdown menu is open, showing two options: 'Create release' and 'Create draft release'. The 'Create release' option is highlighted with a red box.

2. Enter a description for the release, check that the correct artifacts are selected, and then choose **Create**.

Create new release

SampleApp - 1



Click on an environment to change its trigger from automated to manual.

Environments for trigger change from automated to manual. ⓘ





Select the version for the artifact sources for this release

Source alias	Version
SampleApp - 1	20170815.1

Release description

New manual release for Sample App 1

**Create** **Cancel**

3. After a few moments, a banner appears indicating that the new release was created. Choose the link (the name of the release).

SampleApp - 1

Save + Release ⋮ View releases ...



Pipeline Tasks Variables Retention Options History

4. The release summary page opens showing details of the release. In the **Environments** section, you will see the deployment status for the "QA" environment change from "IN PROGRESS" to "SUCCEEDED" and, at that point, a banner appears indicating that the release is now waiting for approval. When a deployment to an environment is pending or has failed, a blue (i) information icon is shown. Point to this to see a pop-up containing the reason.

SampleApp - 1 / Release-2

**Summary** Environments Artifacts Variables General Commits Work item

⟳ | ⌂ Deploy ▾ Save Abandon Send Email

A pre-deployment approval is pending for 'Production' environment [Approve or Reject](#)

**Details**

Manually initiated release with approval before production deployment. ↗

Manually created by Mateo Escobedo 6 minutes ago

30 / 20170815.1 (Build) ↗ master

**Environments**

Environment	Actions	Deployment status	Triggered	Completed
QA	...	SUCCEEDED	6 minutes ago	5 minutes ago
Production	...	NOT DEPLOYED ⓘ	5 minutes ago	

**Issues**

Pending pre-deployment approval

No issues reported in this release.

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 environment 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.

SampleApp - 1 | [Edit](#)

Overview Releases Deleted

⟳ | + Release ▾

Lock	Release	Title	Environments
🔒	Release-2	...	✓ ⓘ
↗	Release-	Production: Pending pre-deployment approval	

- Choose the **Approve or Reject** link to open the approval dialog. Enter a brief note about the approval, and choose **Approve**.

SampleApp - 1 / Release-2

**Summary** Environments Artifacts Variables General Commits Work item

⟳ | ⚡ Deploy ▾ Save Abandon Send Email

A pre-deployment approval is pending for 'Production' environment [Approve or Reject](#)

**Details**

Manually Manually 30 / 2 Approved for deployment to production.

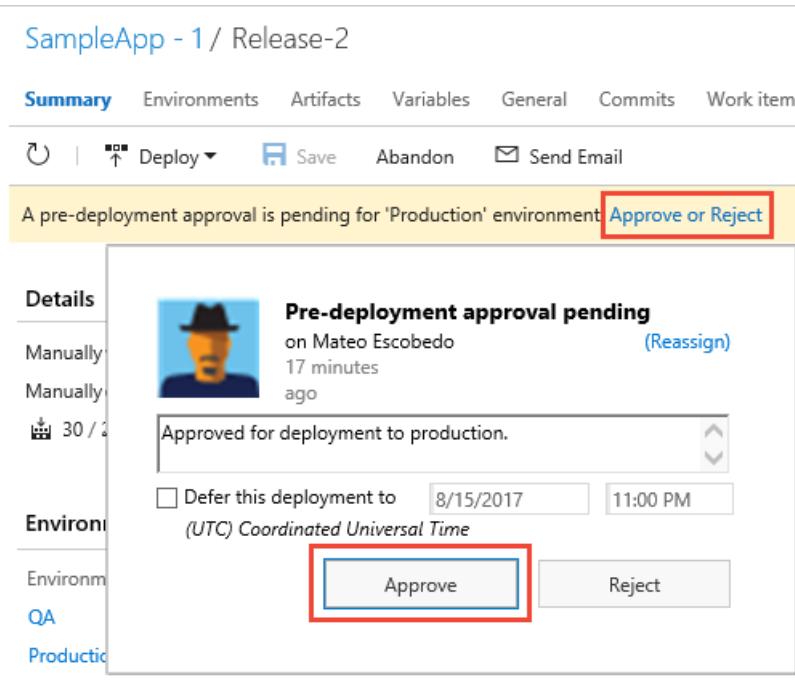
**Environment**

Environment QA Production

**Pre-deployment approval pending**  
on Mateo Escobedo (Reassign)  
17 minutes ago

Defer this deployment to 8/15/2017 (UTC) Coordinated Universal Time

[Approve](#) [Reject](#)



Notice that you can defer a deployment to a specific day and time; for example, a time when you expect the app to be only lightly loaded. You can also reassign the approval to another user. Release administrators can open *any* approval and over-ride it to accept or reject that deployment.

## Monitor and track deployments

In this section, you will see how you can monitor and track deployments - in this example to two Azure App Services websites - from the release you created in the previous section.

1. In the release summary page, choose the **Logs** link. While the deployment is taking place, this page shows the live log from the agent and, in the left pane, an indication of the status of each operation in the deployment process for each environment.

SampleApp - 1 / Release-2

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

Deploy Save Abandon Download all logs as zip Send Email

Step	Action	Log Content
QA	...	Agent queue: Hosted VS2017   Agent: Hosted Agent ***** Starting: Download Artifacts *****
Pre-deployment approval		Downloading artifact Creating artifacts directory: d:\a\r1\a Created artifacts directory: d:\a\r1\a Starting artifacts download...
Phase1	...	Downloading linked artifact Drop of type Build... Ensuring artifact folder d:\a\r1\a\Drop exists and is clean. Preparing to get the list of available artifacts from build Preparing to download artifact: drop Artifact Type: ServerDrop Downloading artifact from file container: #/1161098/drop to t...
Initialize Agent		Download buffer size: 8192 Caching items under 'drop' in the file container...
Initialize Job		Caching complete. (210 ms) Downloading file d:\a\r1\a\Drop\drop\dotnetcore-sample.zip
Download Artifacts		Download complete. 1 placed file(s): 1 downloaded, 0 empty 8 MB downloaded at 6408 KB/sec. Download time: 00:00:01.18905
Deploy Azure App Service		Downloaded linked artifact Drop Finished artifacts download Downloading commits
Post-deployment approval		***** Finishing: Download Artifacts *****
Production	...	***** Starting: Deploy Azure App Service *****
Pre-deployment approval		Task : Azure App Service Deploy Description : Update Azure Web App Service - Web App On Linux
Phase1	...	***** Finishing: Deploy Azure App Service *****
Initialize Agent		*****
Initialize Job		*****
Download Artifacts		*****
Deploy Azure App Service		*****

Choose the icon in the **Action** column for a pre-deployment or post-deployment approval to see details of who approved (or rejected) the deployment, and the message that user provided.

- After the deployment is complete, the entire log file is displayed in the right pane. Select any of the process steps in the left pane to show just the log file contents for that step. This makes it easier to trace and debug individual parts of the overall deployment. Alternatively, download the individual log files, or a zip of all the log files, from the icons and links in the page.

SampleApp - 1 / Release-2

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

⌚ | ⌂ | ⌂ | ⌂ Deploy Save Abandon | [Download all logs as zip](#) | [Send Email](#)

Step	Action	Agent queue: Hosted VS2017   Agent: Hosted Agent	Start Time
QA	...	1 2017-08-15T15:39:50.7248385Z #[section]Starting: Downl...	
Pre-deployment approval	👤	2 2017-08-15T15:39:50.8627576Z Downloading artifact	
Phase1	📄	3 2017-08-15T15:39:50.8727569Z Creating artifacts directo...	
Initialize Agent	📄	4 2017-08-15T15:39:50.8747541Z Created artifacts director...	
Initialize Job	📄	5 2017-08-15T15:39:50.8777550Z Starting artifacts download	
<b>Download Artifacts</b>	📄	6 2017-08-15T15:39:50.8817555Z Downloading linked artifact	
Deploy Azure App Service	📄	7 2017-08-15T15:39:50.8867551Z Ensuring artifact folder c...	
Post-deployment approval	👤	8 2017-08-15T15:39:50.9207555Z Preparing to get the list	
Production	...	9 2017-08-15T15:39:51.8880602Z Preparing to download arti...	
Pre-deployment approval	👤	10 2017-08-15T15:39:51.9110605Z Artifact Type: ServerDrop	
Phase1	📄	11 2017-08-15T15:39:51.9110605Z Downloading artifact from	
Initialize Agent	📄	12 2017-08-15T15:39:51.9120606Z Parallel download limit: 4	
Initialize Job	📄	13 2017-08-15T15:39:51.9120606Z Download buffer size: 8192	
Download Artifacts	📄	14 2017-08-15T15:39:52.4692312Z Caching items under 'drop...	
Deploy Azure App Service	📄	15 2017-08-15T15:39:52.7199471Z Caching complete. (251 ms)	
Post-deployment approval	👤	16 2017-08-15T15:39:52.7379477Z Downloading file d:\a\r1\...	
		17 2017-08-15T15:39:53.7728278Z Download complete.	
		18 2017-08-15T15:39:53.7738279Z 1 placed file(s): 1 downl...	
		19 2017-08-15T15:39:53.7808304Z 8 MB downloaded at 7318 K...	
		20 2017-08-15T15:39:53.7808304Z Downloaded linked artifact	
		21 2017-08-15T15:39:53.7808304Z Finished artifacts download	
		22 2017-08-15T15:39:53.7818278Z Downloading commits	
		23 2017-08-15T15:39:53.7868277Z #[section]Finishing: Down...	
		24	

3. Open the **Summary** tab to see the overall detail of the release. It shows details of the build and the environments it was deployed to - along with the deployment status and other information about the release.

SampleApp - 1 / Release-2

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

⌚ | ⌂ | ⌂ | ⌂ Deploy Save Abandon | [Send Email](#)

**Details**

Manually initiated release with approval before production deployment. [✍](#)

Manually created by Mateo Escobedo just now

30 / 20170815.1 (Build) master

**Environments**

Environment	Actions	Deployment status	Triggered	Completed	Tests
QA	...	SUCCEEDED	just now	just now	No tests
Production	...	SUCCEEDED	just now	just now	No tests

**Issues**

No issues reported in this release.

**Work items**

No associated work items found.

**Tags**

Add...

4. Select each of the environment links to see more details about existing and pending deployments to that specific environment. You can use these pages to verify that the same build was deployed to both environments.

The screenshot shows the 'SampleApp - 1 / Environment: Production' page. At the top, it says 'succeeded'. Below that is an 'Overview' section. The 'Current status' table includes rows for Release (Release-2 succeeded just now), Trigger (Automated - after successful deployment to QA), and Artifacts (30 / 20170815.1 (Build) master). The 'Waiting for deployment' section shows 'No releases queued for deployment'. The 'Recently deployed' section lists a single entry: Release-2 succeeded just now, triggered by 'Automated - aft deployment to QA'. The 'Settings' section contains fields for Pre-deployment approver (Mateo Escobedo), Post-deployment approver (Automated), Trigger (After successful deployment to QA), and Agent queue (Hosted VS2017).

5. Open the deployed production app in your browser. For example, for an Azure App Services website, from the URL [http://\[your-app-name\].azurewebsites.net](http://[your-app-name].azurewebsites.net)

The screenshot shows a web browser window displaying the deployed application. The address bar shows 'http://sampleapp.azurewebsites.net'. The page has a dark header with 'dotnetcore\_sample' and navigation links for 'Home', 'About', and 'Contact'. The main content area has a blue background. It features the 'ASP.NET Core' logo and links for 'Windows', 'Linux', and 'OSX'. A call-to-action button says 'Learn how to build ASP.NET apps that can run anywhere.' with a 'Learn More' link. Below this is a horizontal ellipsis '●○○○'. The page then transitions into a white content area with sections for 'Application uses' (listing MVC, Bower, and Bootstrap), 'How to' (listing Controller and View, User Secrets), and 'Next step' (linking to 'Use approvals and gates to control your deployment').

If you are having problems with a deployment, you can get more information from the log files by [running the release in debug mode](#).

## Next step

[Use approvals and gates to control your deployment](#)

# Use approvals and gates to control your deployment

2/26/2018 • 5 min to read • [Edit Online](#)

By using a combination of manual deployment approvals, gates, and manual intervention within a release definition in Visual Studio Team Services (VSTS) and Team Foundation Server (TFS), you can quickly and easily configure a release pipeline with all the control and auditing capabilities you require.

In this tutorial, you learn about:

- Extending the approval process with gates
- Extending the approval process with manual intervention
- Viewing and monitoring approvals and gates

## Prerequisites

This tutorial extends the tutorial [Define your multi-stage continuous deployment \(CD\) process](#). **You must have completed that tutorial first.**

You'll also need a **work item query** that returns some work items from your VSTS or TFS account. This query is used in the gate you will configure. You can use one of the built-in queries, or create a new one just for this gate to use. For more information, see [Create managed queries with the query editor](#).

In the previous tutorial, you saw a simple use of manual approvals to allow an administrator to confirm that a release is ready to deploy to the production environment. In this tutorial, you'll see some additional and more powerful ways to configure approvals for releases and deployments by using manual intervention and gates. For more information about the ways you can configure approvals for a release, see [Approvals and gates overview](#).

## Configure a gate

First, you will extend the approval process for the release by adding a gate. Gates allow you to configure automated calls to external services, where the results are used to approve or reject a deployment. You can use gates to ensure that the release meets a wide range of criteria, without requiring user intervention.

1. In the **Releases** tab of the **Build & Release** hub, select your release definition and choose **Edit** to open the pipeline editor.

Lock	Pre-deployment conditions	Title	Environments	Build
Unlocked	Production	Release-7	Production	20171108.1 (Build)
Unlocked	Production	Release-6	Production	20171108.1 (Build)

2. Choose the pre-deployment conditions icon for the **Production** environment to open the conditions panel. You can see that there is already an approver configured (this was done in the previous tutorial), but gates are disabled. Enable them by using the switch control in the **Gates** section.

All definitions > SampleApp - 1

Save + Release View releases

Pipeline Tasks Variables Retention Options History

### Pre-deployment conditions

Production

**Triggers** Define the trigger that will start deployment to this environment

**Pre-deployment approvers** Enabled Select the users who can approve or reject deployments to this environment

Approvers [\(1\)](#)

- Mateo Escobedo [X](#) Search users and groups

**Timeout** 1 Days

**Approval policies**

- User requesting a release or deployment should not approve
- Skip approval if the same approver approved the previous environment

**Gates\*** [Disabled](#) Define gates to evaluate before the deployment. [Learn more](#)

- To allow gate functions to initialize and stabilize (it may take some time for them to begin returning accurate results), you configure a delay before the results are evaluated and used to determine if the deployment should be approved or rejected. For this example, so that you can see a result reasonably quickly, set the delay to a short period such as one minute.

Pre-deployment conditions

Production

**Triggers** Define the trigger that will start deployment to this environment

**Pre-deployment approvers** Enabled Select the users who can approve or reject deployments to this environment

**Gates\*** Enabled Define gates to evaluate before the deployment. [Learn more](#)

**Delay before evaluation \*** 1 Minutes

**Deployment gates** [Add](#)

- Choose **+ Add** and select the **Query Work Items** gate.

Gates\* ^

Define gates to evaluate before the deployment. [Learn more](#)

Enabled

Delay before evaluation \* ⓘ

1 Minutes ⓘ

Deployment gates ⓘ

+ Add

Azure Monitor  
Observe the configured Azure monitor rules for active alerts.

Invoke Azure Function  
Invoke Azure Function as a part of your process.

Invoke REST API  
Invoke REST API as a part of your process.

Publish To Azure Service Bus  
Sends a message to azure service bus using a service connection (no agent required).

Query Work Items  
Executes a work item query and checks for the number of items returned.

- Configure the gate by selecting an existing work item query. You can use one of the built-in VSTS and TFS queries, or [create your own query](#). Depending on how many work items you expect it to return, set the maximum and minimum thresholds (run the query in the **Work** hub if you're not sure what to expect).

Gates\* ^

Define gates to evaluate before the deployment. [Learn more](#)

Enabled

Delay before evaluation \* ⓘ

1 Minutes ⓘ

Deployment gates ⓘ

+ Add

Query Work Items

Enabled

Query Work Items (Preview) ⓘ

Display name \*

Query Work Items

Query \* ⓘ

Active Bugs

Maximum Threshold \* ⓘ

10

Advanced ^

Minimum Threshold \* ⓘ

0

Options for all gates ⓘ

You'll need to open the **Advanced** section to see the **Maximum Threshold** setting. For more details about the gate arguments, see [Work Item Query task](#).

- Open the **Options for all gates** section and specify the timeout and the sampling interval. For this example, choose short periods so that you can see the results reasonably quickly. The minimum values you can

specify are 6 minutes timeout and 5 minutes sampling interval.

Advanced ^

Minimum Threshold \* ⓘ

0

Options for all gates ^

Timeout \* ⓘ

6 Minutes

Sampling interval \* ⓘ

5 Minutes

Execution order of approvals and gates ⓘ

Manual approvals before gates

Manual approvals after all gates succeed

Manual approvals after gates - always

The sampling interval and timeout work together so that the gates will call their functions at suitable intervals, and reject the deployment if they don't all succeed during the same sampling interval and within the timeout period. For more details, see [Gates](#).

7. Save your release definition.

All definitions > SampleApp - 1

Save

For more information about using other types of approval gates, see [Approvals and gates](#).

## Configure a manual intervention

Sometimes, you may need to introduce manual intervention into a release pipeline. For example, there may be tasks that cannot be accomplished automatically such as confirming network conditions are appropriate, or that specific hardware or software is in place, before you approve a deployment. You can do this by using the **Manual Intervention** task in your pipeline.

1. In the release pipeline editor, open the **Tasks** editor for the **QA** environment.

All definitions > SampleApp - 1

Pipeline Tasks Variables Retention Options History

QA

Production

Artifacts | + Add Environments | + Add

2. Choose the ellipses (...) in the **QA** deployment process bar and then choose **Add agentless phase**.

All definitions > SampleApp - 1

Pipeline Tasks Variables Retention Options History

QA Deployment process

Agent phase Run on agent

Azure App Service Depl... Azure App Service Deploy

...

Add agent phase

+ Add deployment group phase

+ Add agentless phase

Learn more about phases

The screenshot shows the 'Tasks' tab selected in the pipeline editor. A context menu is open over the 'Agentless phase' row, with the 'Add agentless phase' option highlighted by a red box.

Several tasks, including the **Manual Intervention** task, can be used only in an **agentless** phase.

3. Choose + in the **Agentless phase** bar and add a **Manual Intervention** task to the phase.

All definitions > SampleApp - 1

Save Release

Pipeline Tasks Variables Retention Options History

QA Deployment process

Agentless phase Run on server

Agent phase Run on agent

Azure App Service Depl... Azure App Service Deploy

+

Add tasks

Don't see what you need? Check out our Marketplace.

All Utility Deploy

Azure Monitor Observe the configured Azure monitor rules for active alerts.

Delay Delay further execution of the workflow by a fixed time.

Invoke Azure Function Invoke Azure Function as a part of your process.

Invoke REST API Invoke REST API as a part of your process.

Manual Intervention Pause deployment and wait for intervention

by Microsoft Corporation Add

Learn more

Publish To Azure Service Bus Sends a message to azure service bus using a service connection (no agent required).

The screenshot shows the 'Tasks' tab selected. An 'Agentless phase' row is selected, indicated by a red box around its '+' icon. A modal window titled 'Add tasks' is open, listing various tasks. The 'Manual Intervention' task is visible, and its 'Add' button is highlighted with a red box.

4. Configure the task by entering a message (the **Instructions**) to display when it executes and pauses the release process.

The screenshot shows the 'Tasks' tab of a release definition named 'SampleApp - 1'. On the left, there's a list of tasks: 'QA Deployment process', 'Agentless phase Run on server', 'Manual Intervention Manual Intervention' (which is highlighted with a red box), 'Agent phase Run on agent', and 'Azure App Service Deploy... Azure App Service Deploy'. The 'Manual Intervention' task has a checkmark and three vertical dots. To the right, the configuration for this task is shown. It includes a 'Version' dropdown set to '8.\*', a 'Display name' field containing 'Manual Intervention', and an 'Instructions' field with the text 'Ensure QA database updates have completed before continuing deployment'. Below these are sections for 'Notify users' (listing 'Mateo Escobedo') and 'On timeout' (with options 'Reject' and 'Resume').

Notice that you can specify a list of users who will receive a notification that the deployment is waiting for manual approval. You can also specify a timeout and the action (approve or reject) that will occur if there is no user response within the timeout period. For more details, see [Manual Intervention task](#).

5. Save the release definition and start a new release.

The screenshot shows the 'All definitions > SampleApp - 1' page. The top navigation bar includes 'Save', 'Release', and 'View releases'. A dropdown menu under 'Release' is open, showing 'Create release' and 'Create draft release', both highlighted with a red box. The main area displays the 'QA Deployment process' and the 'Manual Intervention' task.

## View the logs for approvals

You typically need to validate and audit a release and the associated deployments after it has completed, or even during the deployment process. This is useful when debugging a problematic deployment, or when checking when and by whom approvals were granted. The comprehensive logging capabilities provide this information.

1. Open the release summary for the release you just created. You can do this by choosing the link in the information bar in the release editor after you create the release, or directly from the **Releases** tab of the **Build & Release** hub.

All release definitions

Overview Releases Deleted

Release-2 SampleApp - 1

Release-1

**Open in new tab**

Start

Retain indefinitely

Abandon

Delete

2. Open the **Logs** page. You'll see a live log and status for each step in the release process. After the release is complete, choose the icon in the **Action** column for the **Manual Intervention** task to see details of who approved, when the approval occurred, and the message entered by the approver.

SampleApp - 1 / Release-7

Logs

QA

Pre-deployment approval

Agentless phase

Manual Intervention

Agent phase

Initialize Job

Initialize Agent

Download Artifacts

Post-deployment approval

Production

Pre-deployment approval

Pre-deployment gates

Query Work Items

Agent phase

Initialize Agent

Initialize Job

Download Artifacts

Post-deployment approval

**Manual Intervention is resumed**  
by Mateo Escobedo  
3 hours ago

Instructions  
Ensure QA database updates have completed before continuing deployment

Comments  
Validated with QA team

3. The release also required an approval to start deployment to the production environment. Choose the icon in the **Action** column for the **Pre-deployment** condition. Again, you see details of the approval.

## SampleApp - 1 / Release-7

Summary Environments Artifacts Variables General Commits Work items Tests Logs History

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

Step	Action	Start Time: 09/11/2017 18:24
QA	...	1 The pre-deployment approval has been Approved. 2 Please click on the approval icon in the left pane
Pre-deployment approval	🔍	
Agentless phase	🔗	
Manual Intervention	🔍	
Agent phase	🔗	
Initialize Job	🔗	
Initialize Agent	🔗	
Download Artifacts	🔗	
Post-deployment approval	🔍	
Production	...	
Pre-deployment approval	🔍	
Pre-deployment gates		
Query Work Items		
Agent phase		
Initialize Agent		
Initialize Job		
Download Artifacts		
Post-deployment approval	🔍	



**Pre-deployment approval approved**  
by Mateo Escobedo  
3 hours ago

QA deployment success, ready for production

- Finally, the release was approved by a gate, which validated the results of a work item query (which would be used to ensure the work required was complete and any bugs reported had been resolved). Choose the icon in the **Action** column for the **Query Work Items** gate. The information panel shows the result at each sample interval when the gate executed the work item query.

## SampleApp - 1 / Release-7

Summary Environments Artifacts Variables General Commits Work items Tests Logs History

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

Step Action

### QA

Pre-deployment approval



Agentless phase



Manual Intervention



Agent phase



Initialize Job



Initialize Agent



Download Artifacts



Post-deployment approval



### Production

Pre-deployment approval



Pre-deployment gates

Query Work Items

Agent phase



Initialize Agent



Initialize Job



Download Artifacts



Post-deployment approval



### Query Work Items

Status	Sample time	Duration	Logs
--------	-------------	----------	------

✓	09/11/2017 18:25	0 seconds	🔗
---	------------------	-----------	---

✓	09/11/2017 18:30	0 seconds	🔗
---	------------------	-----------	---

Showing latest 2 of 2 samples. Download all logs to get results for all samples.

Altogether, by using a combination of manual approvals, approval gates, and the manual intervention task, you've seen how can configure a release pipeline with all the control and auditing capabilities you may require.

## Next step

[Deploy to IIS web servers on Windows](#)

# Create a CI/CD pipeline for your existing code with the Azure DevOps Project

2/13/2018 • 7 min to read • [Edit Online](#)

The Azure DevOps Project presents a simplified experience where you can bring your existing code and Git repository, or choose from one of the sample applications to create a continuous integration (CI) and continuous delivery (CD) pipeline to Azure.

You will:

- Create an Azure DevOps project
- Configure access to your GitHub repository and choose a framework
- Configure VSTS and an Azure subscription
- Commit changes to GitHub and automatically deploy to Azure
- Examine the VSTS CI/CD pipeline
- Configure Azure Application Insights monitoring

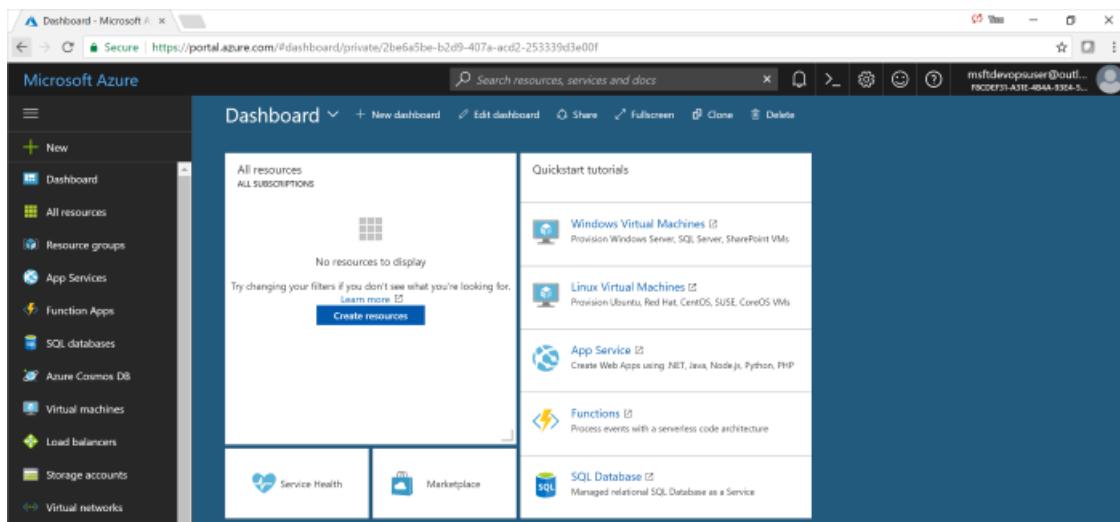
## Prerequisites

- An Azure subscription. You can get one free through [Visual Studio Dev Essentials](#).
- Access to a GitHub or external Git repository that contains .NET, Java, PHP, Node, Python, or static web code.

## Sign in to the Azure portal

The Azure DevOps Project creates a CI/CD pipeline in VSTS. You can create a **new VSTS account** or use an **existing account**. The Azure DevOps Project also creates **Azure resources** in the **Azure subscription** of your choice.

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



3. Select **Bring your own code**. When you're done, choose **Next**.

## Configure access to your GitHub repository and choose a framework

1. Select **GitHub**. Optionally, you can choose an **external Git repository**. Choose your **repository** and

**branch** that contains your application.

2. Select your **web framework**. When you're done, choose **Next**.

Code Repository    Application/Framework    Service    Create

Tell us more about the code

\* Web Application framework ⓘ

ASP.NET

ASP.NET

ASP.NET Core

Node.js

PHP

Python

Static Webapp

3. The application framework, which you chose on the previous steps, dictates the type of Azure service deployment target available here. Select the **target service** of your choice. When you're done, choose **Next**.

## Configure VSTS and an Azure subscription

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



**Almost there!**  
Ready to deploy an Express.js web app to a new Web App on Windows.

## Visual Studio Team Services

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

\* Account

Create new  Use existing

devopstutorial

\* Name

nodesample

## Azure

[Change](#)

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

\* Subscription

Pay-As-You-Go

\* App Service Location

South Central US

\* Name

nodesamplesite



.azurewebsites.net

Pricing tier: S1 Standard

[Previous](#)

[Done](#)

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

The screenshot shows the Azure DevOps project dashboard. On the left, there's a sidebar with 'CI/CD Pipeline' and 'Repository' sections. The 'CI/CD Pipeline' section shows a flow from 'Code' (dotnetcore-sample) through 'Build' (byocsamplesite) to 'Production' (byocsamplesite). The 'Build' step is currently 'In Progress'. The 'Repository' section shows a GitHub repository 'dotnetcore-sample'. On the right, there are three main sections: 'Azure resources', 'Application endpoint' (http://byocsamplesite.azurewebsites.net), and 'Application Insights'. The 'Application Insights' section displays a timeline from 1 AM to 6 AM with a single 'SERVER REQUEST' event.

The Azure DevOps project automatically configures a CI build and release trigger. Your code remains in your GitHub or other external repository.

## Commit changes to GitHub and automatically deploy to Azure

You're now ready to collaborate with a team on your app with a CI/CD process that automatically deploys your latest work to your web site. Each change to the GitHub repo initiates a build in VSTS, and a VSTS Release Management definition executes a deployment to Azure.

1. Make a change to your application, and **commit** the change to your GitHub repository.
2. In a few moments, a build initiates in VSTS. You can monitor the build status with the DevOps project dashboard or in the browser with your VSTS account.
3. Once the build completes, **refresh your application** in the browser to verify you see your changes.

## Examine the VSTS CI/CD pipeline

The Azure DevOps project automatically configured a full VSTS CI/CD pipeline in your VSTS account. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release definitions.

1. Select **Build Pipelines** from the **top** of the Azure DevOps project dashboard. This link opens a browser tab and opens the VSTS build definition for your new project.
2. Move the mouse cursor to the right of the build definition next to the **Status** field. Select the **ellipsis** that appears. This action opens a menu where you can perform several activities such as queue a new build, pause a build, and edit the build definition.
3. Select **Edit**.
4. From this view, **examine the various tasks** for your build definition. The build performs various tasks such as fetching sources from the Git repository, restoring dependencies, and publishing outputs used for deployments.
5. At the top of the build definition, select the **build definition name**.
6. Change the **name** of your build definition to something more descriptive. Select **Save & queue**, then select

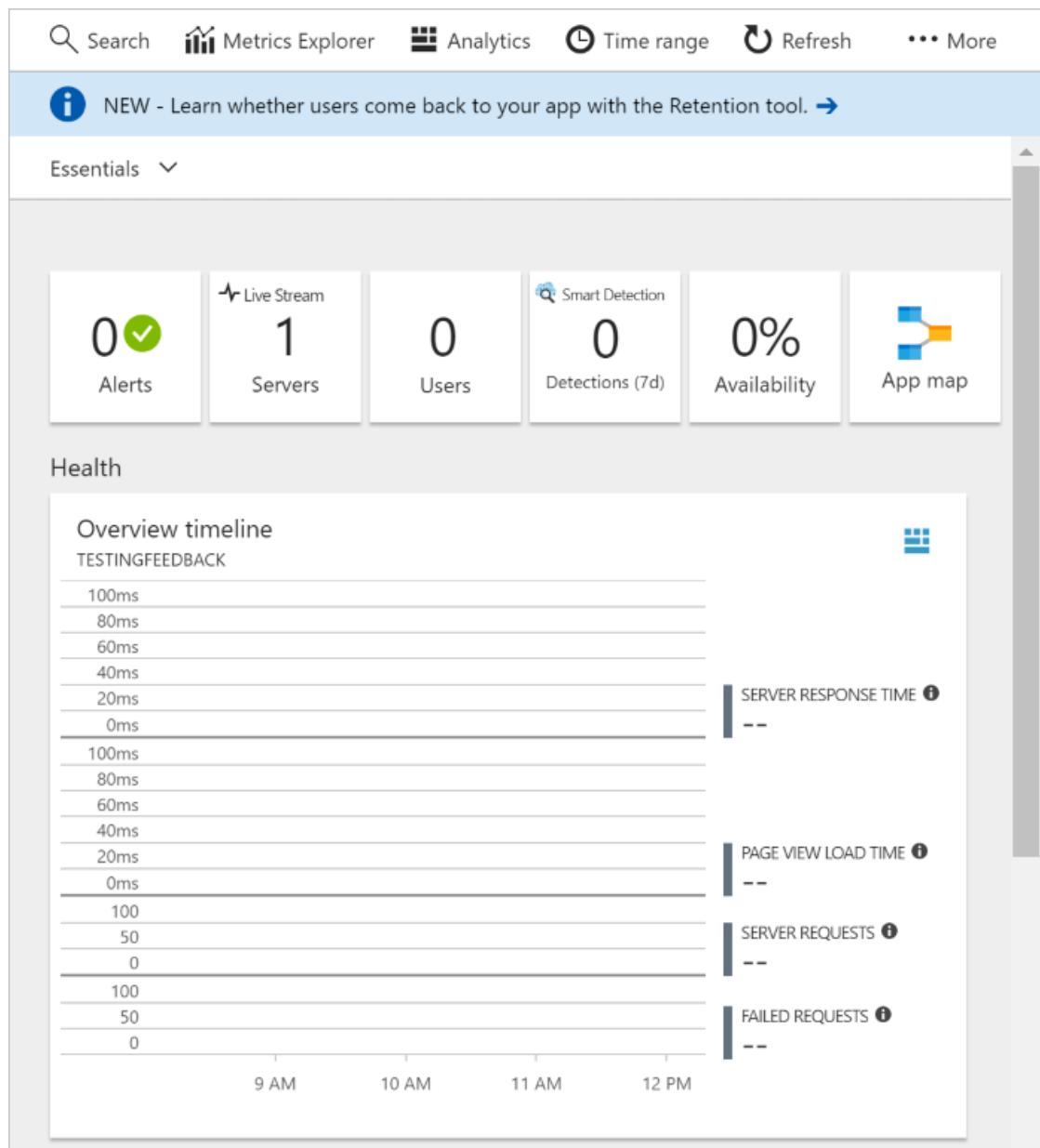
## **Save.**

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

## Configure Azure Application Insights monitoring

With Azure Application insights, you can easily monitor your application's performance and usage. The Azure DevOps project automatically configured an Application Insights resource for your application. You can further configure various alerts and monitoring capabilities as needed.

1. Navigate to the **Azure DevOps Project** dashboard in the Azure portal. On the bottom-right of the dashboard, choose the **Application Insights** link for your app.
2. The **Application Insights** blade opens in the Azure portal. This view contains usage, performance, and availability monitoring information for your app.



3. Select **Time range**, and then choose **Last hour**. Select **Update** to filter the results. You now see all activity from the last 60 minutes. Select the **x** to exit time range.
4. Select **Alerts**, then select **+ Add metric alert**.
5. Enter a **Name** for the alert.
6. Select the drop-down for **Source Alter on**. Choose your **App Service resource**.
7. The default alert is for a **server response time greater than 1 second**. Select the **Metric** drop-down to examine the various alert metrics. You can easily configure a variety of alerts to improve the monitoring capabilities of your app.
8. Select the check-box for **Notify via Email owners, contributors, and readers**. Optionally, you can perform additional actions when an alert fires by executing an Azure logic app.
9. Choose **Ok** to create the alert. In a few moments, the alert appears as active on the dashboard. **Exit** the Alerts area, and navigate back to the **Application Insights blade**.
10. Select **Availability**, then select **+ Add test**.
11. Enter a **Test name**, then choose **Create**. A simple ping test is created to verify the availability of your application. After a few minutes, test results are available, and the Application Insights dashboard displays an availability status.

## Clean up resources

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

## Next steps

When you configured your CI/CD process in this tutorial, a build and release definition were automatically created in your VSTS project. You can modify these build and release definitions to meet the needs of your team. You learned how to:

- Create an Azure DevOps project
- Configure access to your GitHub repository and choose a framework
- Configure VSTS and an Azure subscription
- Commit changes to GitHub and automatically deploy to Azure
- Examine the VSTS CI/CD pipeline
- Configure Azure Application Insights monitoring

To learn more about the VSTS pipeline see this tutorial:

[Customize CD process](#)

# Build and Release Agents

12/19/2017 • 11 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)**

To build your code or deploy your software you need at least one agent. As you add more code and people, you'll eventually need more.

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

## Hosted agents

If your build and release definitions are in VSTS, then you've got a convenient option to build and deploy using a **hosted agent**. When you use a hosted agent, we take care of the maintenance and upgrades. So for many teams this is the simplest way to build and deploy. You can try it first and see if it works for your build or deployment. If not, you can set up a private agent.

### TIP

You can try a hosted agent for no charge. If your build or release doesn't succeed, the issues will be reported in the logs.

[Learn more about hosted agents.](#)

## Private agents

An agent that you set up and manage on your own to run build and deployment jobs is a **private agent**. You can use private agents in VSTS or Team Foundation Server (TFS). Private agents give you more control to install dependent software needed for your builds and deployments.

You can install the agent on Windows, Linux, or macOS machines. You can also install an agent on a Linux Docker container.

After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.

### Install and connect to VSTS and TFS 2017 and newer

#### TIP

Is your code in VSTS? If so, before you install an agent you might want to see if the hosted pool will work for you. In many cases this is the simplest way to get going. [Give it a try.](#)

- [Windows agent v2](#)
- [macOS agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [RedHat agent](#)

### Install and connect to TFS 2015

- [Windows agent v1](#)
- [macOS agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [RedHat agent](#)

### Concurrent pipelines for private agents

You might need more concurrent pipelines to use multiple agents at the same time:

- [Concurrent pipelines in VSTS](#)
- [Concurrent pipelines in TFS](#)

## Capabilities

Every agent has a set of capabilities that indicate what it can do. Capabilities are name-value pairs that are either automatically discovered by the agent software, in which case they are called **system capabilities**, or those that you define, in which case they are called **user capabilities**.

The agent software automatically determines various system capabilities such as the name of the machine, type of operating system, and versions of certain software installed on the machine. Also, environment variables defined in the machine automatically appear in the list of system capabilities.

When you author a build or release definition, or when you queue a build or deployment, you specify certain **demands** of the agent. The system sends the job only to agents that have capabilities matching the demands [specified in the definition](#). As a result, agent capabilities allow you to direct builds and deployments to specific agents.

You can view the system capabilities of an agent, and manage its user capabilities by navigating to the **Agent pools** hub and selecting the **Capabilities** tab for the desired agent.

- VSTS: [https://{{your\\_account}}.visualstudio.com/\\_admin/\\_AgentPool](https://{{your_account}}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{{your\\_server}}/tfs/DefaultCollection/\\_admin/\\_AgentPool](https://{{your_server}}/tfs/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{{your\\_server}}:8080/tfs/\\_admin/\\_AgentPool](http://{{your_server}}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

#### TIP

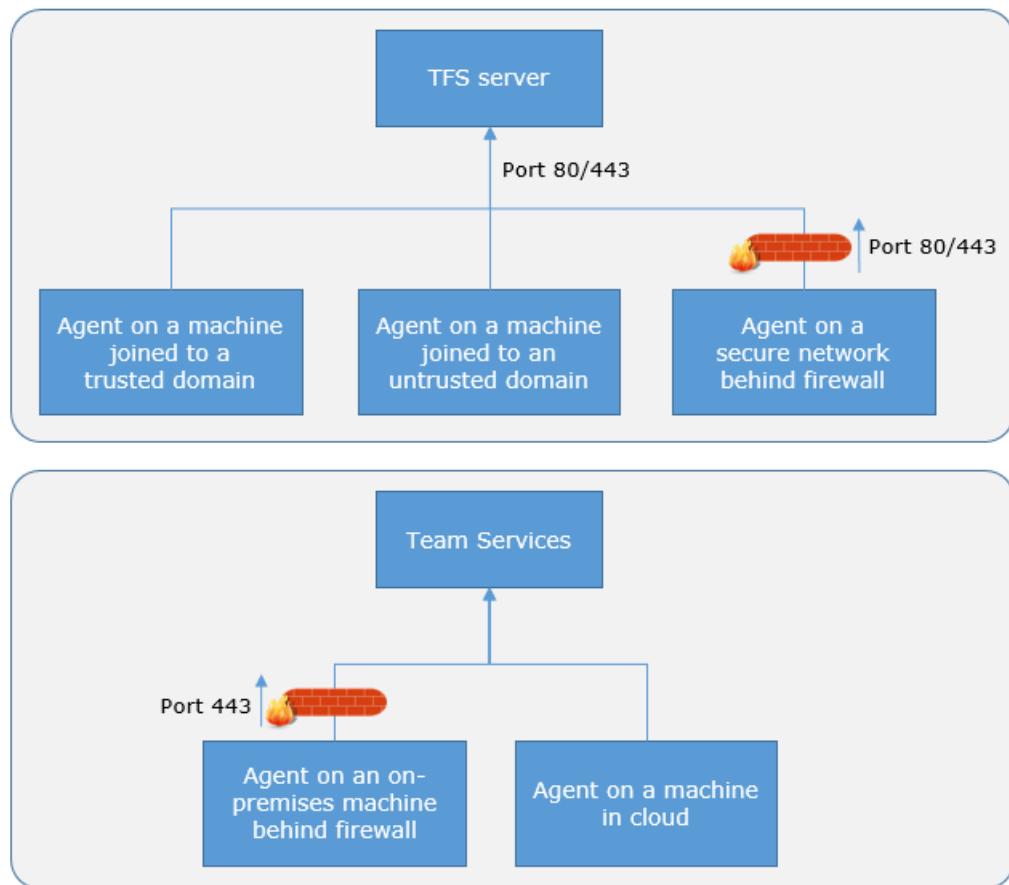
After you install new software on a agent, you must restart the agent for the new capability to show up.

## Communication

### Communication with VSTS or TFS

#### VSTS or TFS 2017 and newer

The agent communicates with VSTS 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 VSTS 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 VSTS or TFS.

1. The user registers an agent with VSTS or TFS by adding it to an [agent pool](#). You need to be an [agent pool administrator](#) to register an agent in that agent pool. The identity of agent pool administrator is needed only at the time of registration and is not persisted on the agent, nor is used in any further communication between the agent and VSTS or TFS. Once the registration is complete, the agent downloads a *listener OAuth token* and uses it to listen to the job queue.
2. Periodically, the agent checks to see if a new job request has been posted for it in the job queue in TFS/VSTS. When a job is available, agent downloads the job as well as a *job-specific OAuth token*. This token is generated by TFS/VSTS for the scoped identity [specified in the definition](#). That token is short lived and is used by agent to access (e.g., source code) or modify resources (e.g., upload test results) on VSTS or TFS within that job.
3. Once the job is completed, 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 TFS/VSTS are secured using asymmetric encryption. Each agent has a public-private key pair, and the public key is exchanged with the server during registration. Server uses the public key to encrypt the payload of the job before sending it to the agent. The agent decrypts the job content using its private key. This is how secrets stored in build definitions, release definitions, or variable groups are secured as they are exchanged with the agent.

#### TFS 2015

In TFS 2015:

- 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 macOS and Linux) are used to initiate communication with TFS. The agent uses these credentials to listening to the job

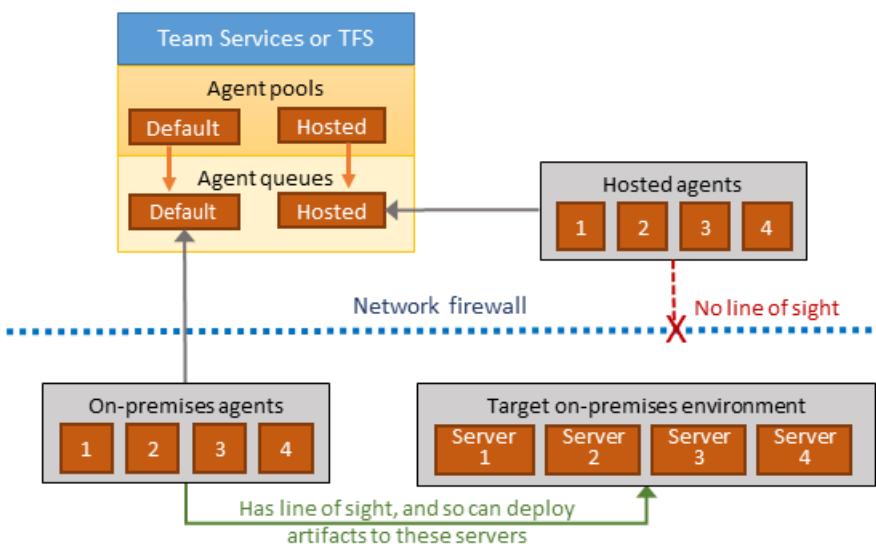
queue.

- 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 hosted pool, by default, has connectivity to Windows Azure websites and Windows servers running in Azure.

If your on-premises environments do not have connectivity to the hosted pool (which is typically the case due to intermediate firewalls), you'll need to manually configure a private 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 VSTS 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. Your agent can authenticate to VSTS or TFS using one of the following methods:

- **Personal Access Token (PAT):** [Generate](#) and use a PAT to connect an agent with VSTS or TFS 2017 and newer. PAT is the only scheme that works with VSTS.
- **Integrated:** Connect a Windows agent to TFS using the credentials of the signed-in user via a Windows authentication scheme such as NTLM or Kerberos.
- **Negotiate:** Connect to TFS as a user other than the signed-in user via a Windows authentication scheme such as NTLM or Kerberos.
- **Alternate:** Connect to TFS using Basic authentication. To use this method you'll first need to [configure HTTPS on TFS](#).

## Interactive vs. service

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

For example, to run tasks that use Windows authentication to access an external service, you must run

the agent using an account that has access to that service. However, if you are running UI tests such as Selenium or Coded UI tests that require a browser, the browser is launched in the context of the agent account.

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

1. **As a service.** You can leverage the service manager of the operating system to manage the lifecycle of the agent. In addition, the experience for auto-upgrading the agent is better when it is run as a service.
2. **As an interactive process with auto-logon enabled.** In some cases, you might need to run the agent interactively for production use - such as to run UI tests. When the agent is configured to run in this mode, the screen saver is also disabled. Some domain policies may prevent you from enabling auto-logon or disabling the screen saver. In such cases, you may need to seek an exemption from the domain policy, or run the agent on a workgroup computer where the domain policies do not apply.

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

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

## Agent version and upgrades

We update the agent software every few weeks in VSTS, and with every update in TFS. We indicate the agent version in the format `{major}.{minor}`. For instance, if the agent version is `2.1`, then the major version is 2 and the minor version is 1. When a newer version of the agent is only different in minor version, it is automatically upgraded by VSTS or TFS. This upgrade happens when one of the tasks requires a newer version of the agent.

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

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

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

## Q&A

### Do private agents have any performance advantages over hosted agents?

In many cases, yes. Specifically:

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

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

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

You might also run into problems if concurrent build processes 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.

### How do I make sure I have the latest v2 agent version?

1. Go to the *Agent pools* control panel tab:

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool that contains the agent.
3. Make sure the agent is enabled.
4. Click **Agents**.
5. Click **Capabilities**.
6. Look for the `Agent.Version` capability.

You can check this value against the latest published agent version. See [VSTS Build and Release Agent](#) and check the page for the highest version number listed.

7. Each agent automatically updates itself when it runs a task that requires a newer version of the agent. But if you want to manually update some agents, right-click the pool, and then click **Update all agents**.

# Agent pools and queues

2/16/2018 • 7 min to read • [Edit Online](#)

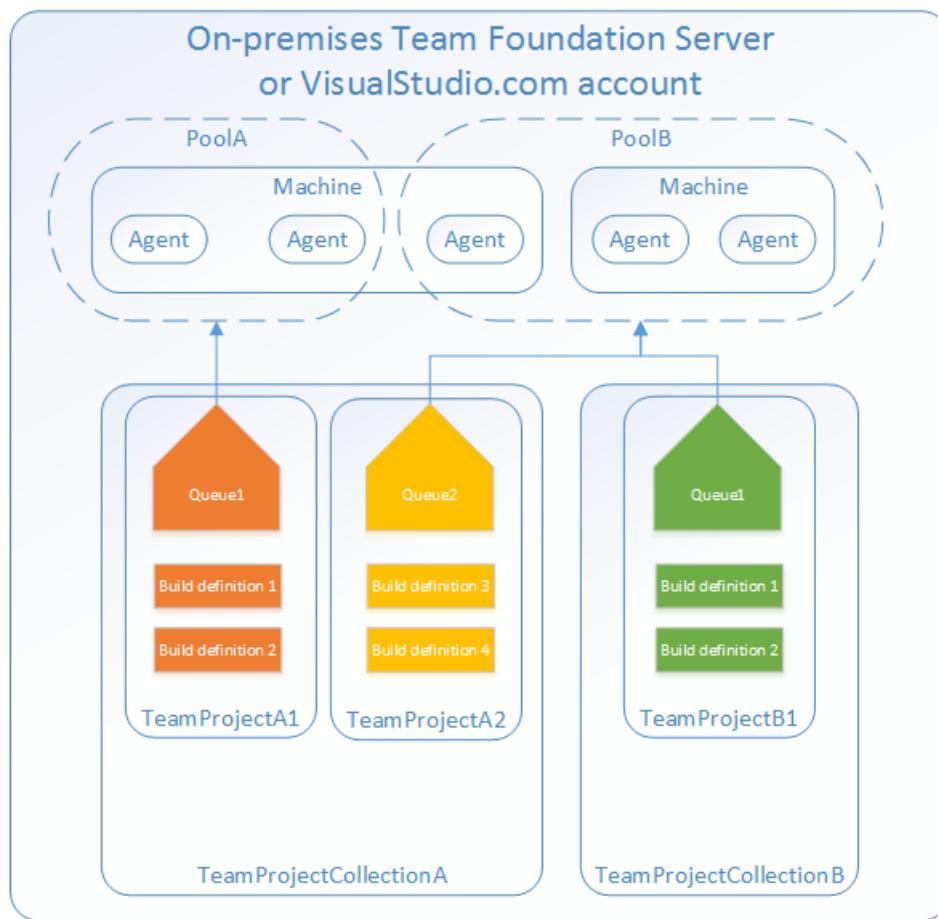
## VSTS | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

Instead of managing each [agent](#) individually, you organize agents into **agent pools**. An agent pool defines the sharing boundary for all agents in that pool. In TFS, pools are scoped across all of your Team Foundation Server (TFS); so you can share an agent pool across team project collections and team projects. In VSTS, agent pools are scoped to the VSTS account; so you can share an agent pool across team projects.

An **agent queue** provides access to an agent pool. When you create a build or release definition, you specify which queue it uses. Queues are scoped to your team project in TFS 2017 and newer and in VSTS, so you can only use them across build and release definitions within a team project.

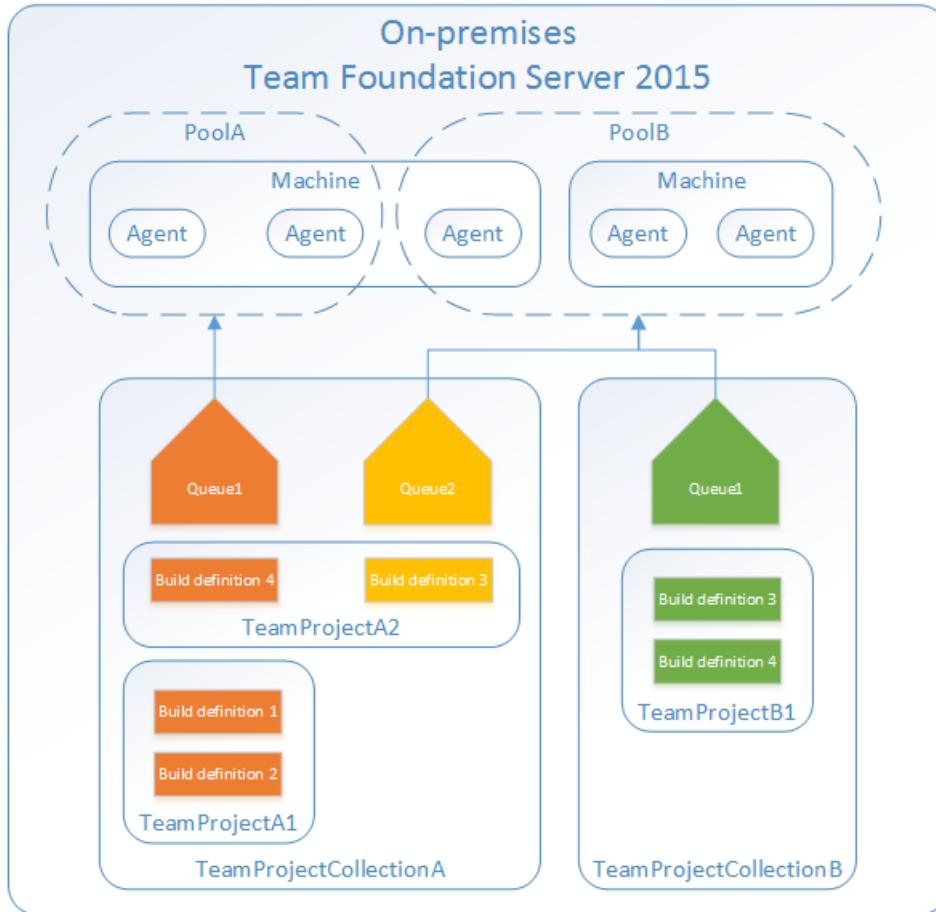
To share an agent pool with multiple team projects, you create an agent queue pointing to that pool in each of those team projects. While multiple queues across team projects can use the same agent pool, multiple queues within a team project cannot use the same pool. Also, each queue can use only one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

### VSTS and TFS 2017 and newer



### TFS 2015

In TFS 2015 agent queues are scoped to team project collections.



You create and manage pools from the Agent pools tab.

- VSTS: [https://{{your\\_account}}.visualstudio.com/\\_admin/\\_AgentPool](https://{{your_account}}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{{your\\_server}}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{{your_server}}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{{your\\_server}}:8080/tfs/\\_admin/\\_AgentPool](http://{{your_server}}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

You create and manage your queues from the Agent queues tab.

- VSTS: [https://{{your\\_account}}.visualstudio.com/{{project-name}}/\\_admin/\\_AgentQueue](https://{{your_account}}.visualstudio.com/{{project-name}}/_admin/_AgentQueue)
- TFS 2017 and newer:  
[https://{{your\\_server}}/tfss/{{collection-name}}/{{project-name}}/\\_admin/\\_AgentQueue](https://{{your_server}}/tfss/{{collection-name}}/{{project-name}}/_admin/_AgentQueue)
- TFS 2015 RTM: [http://{{your\\_server}}:8080/tfs/\\_admin/\\_buildQueue](http://{{your_server}}:8080/tfs/_admin/_buildQueue)
- TFS 2015.3: [http://{{your\\_server}}:8080/tfs/{{collection-name}}/\\_admin/\\_AgentQueue](http://{{your_server}}:8080/tfs/{{collection-name}}/_admin/_AgentQueue)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

## Default agent pools

We provide the following agent pools by default:

- **Default** pool: Use it to register [private agents](#) that you've set up.
- **Hosted** pool (VSTS only): Contains at least one free hosted agent, and also any [hosted agents you've purchased](#). The **Hosted** pool is the built-in pool that is a collection of hosted agents.

Machines in this pool have Visual Studio 2010, Visual Studio 2012, Visual Studio 2013, and Visual Studio 2015 installed on Windows Server 2012 R2 operating system. For a complete list of software installed on hosted agents, see [Hosted agents](#).

- **Hosted VS2017** pool (VSTS only): The **Hosted VS2017** pool is another built-in pool in VSTS. Machines in this pool have Visual Studio 2017 installed on Windows Server 2016 operating system. For a complete list of software installed on these machines, see [Hosted agents](#).
- **Hosted Linux** pool (VSTS only): Enables you to build and release on Linux machines without having to configure a private agent. The agents in this pool run on an Ubuntu Linux host inside the **vsts-agent-docker** container.
- **Hosted macOS Preview** pool (VSTS only): Enables you to build and release on Mac machines without having to configure a private agent. This option affects where your data is stored. [Learn more](#)

Each of these hosted pools is exposed to each team project through a corresponding hosted queue. By default, all contributors in a team project are members of the **User** role on each hosted queue. This allows every contributor in a team project to author and run build and release definitions using hosted queues.

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

## Creating agent pools and queues

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

- You're a member of a team 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 Queues** with the **Administrator** role. Next create a **New queue** in your team project and select the option to **Create a new pool**. As a result, both a queue and a 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 team projects. First make sure you're a member of a group in **All Pools** with the **Administrator** role. Next create a **New pool** and select the option to **Auto-provision queues in all projects** while creating the pool. This setting ensures all team projects have a queue to access the pool. The system creates a queue 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 team projects, but not all of them. First create an agent queue in one of the projects and select the option to **Create a new pool** while creating that queue. Next, go to each of the other team projects, and create a queue in each of them while selecting the option to **Use an existing pool**. Finally, [install](#) and configure agents to be part of the shared agent pool.

## Security of agent pools and queues

Understanding how security works for agent pools and queues helps you control sharing and use of agents.

### VSTS and TFS 2017 and newer

In VSTS and TFS 2017 and newer, **roles** are defined on each agent pool, and **membership** in these roles governs what operations you can perform on an agent pool.

ROLE ON AN AGENT POOL	PURPOSE
Reader	Members of this role can view the pool as well as agents. You typically use this to add operators that are responsible for monitoring the agents and their health.

ROLE ON AN AGENT POOL	PURPOSE
Service Account	Members of this role can use the pool to create an agent queue in a team project. If you follow the guidelines above for creating new pools and queues, you typically do not have to add any members here.
Administrator	In addition to all the above permissions, members of this role can register or unregister agents from the pool. They can also use the agent pool when creating an agent queue in a team project. Finally, they can also manage membership for all roles of the pool. The user that created the pool is automatically added to the Administrator role for that pool.

The **All Pools** node in the Agent Pools tab is used to control the security of *all* agent pools. Role memberships for individual agent pools are automatically inherited from those of the 'All Pools' node. By default, TFS administrators are also administrators of the 'All Pools' node.

Roles are also defined on each agent queue, and memberships in these roles govern what operations you can perform on an agent queue.

ROLE ON AN AGENT QUEUE	PURPOSE
Reader	Members of this role can view the queue. You typically use this to add operators that are responsible for monitoring the build and deployment jobs in that queue.
User	Members of this role can use the queue when authoring build or release definitions.
Administrator	In addition to all the above operations, members of this role can manage membership for all roles of the queue. User that created the queue is automatically added to the Administrator role for that queue.

The **All Queues** node in the Agent Queues tab is used to control the security of *all* agent queues in a team project. Role memberships for individual agent queues are automatically inherited from those of the 'All Queues' node. By default, the following groups are added to the Administrator role of 'All Queues': 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 account-level Agent Pool Administrators group to grant them permission manage all the agent pools. This enables people to create new pools and modify all existing pools. Members of Team Foundation Administrators group can also perform all these operations.

Users in the **Agent Pool Service Accounts** group have permission to listen to the message queue for the specific pool to receive work. In most cases you should not have to manage members of this group. The agent registration process takes care of it for you. The service account you specify for the agent (commonly Network Service) is automatically added when you register the agent.

## Q&A

### I'm trying to create a queue that uses an existing pool, but the controls are grayed out. Why?

On the Create Queue dialog box, you can't use an existing pool if it is already referenced by another queue. Each pool can be referenced by only one queue within a given team project collection.

### I can't select the Hosted queue and I can't queue my build. How do I fix this?

Ask the owner of your VSTS account to grant you permission to use the queue. See [Security of agent pools and queues](#).

### I need more hosted build resources. What can I do?

A: The hosted pool provides all VSTS accounts with a single hosted build agent and a limited number of free build minutes each month. If you need more hosted build resources, or need to run more than one build concurrently, then you can either:

- [Deploy your own on-premises build agents](#).
- [Buy additional hosted pipelines](#).

# Hosted agents

1/26/2018 • 4 min to read • [Edit Online](#)

## VSTS

If your build and release definitions are in VSTS, then you've got a convenient option to build and deploy using a **hosted agent**. When you use a hosted agent, we take care of the maintenance and upgrades. So for many teams this is the simplest way to build and deploy. You can try it first and see if it works for your build or deployment. If not, you can set up a private agent.

### TIP

You can try a hosted agent for no charge. If your build or release doesn't succeed, the issues will be reported in the logs.

## Use a hosted agent

We provide hosted agents to you in our hosted pools. To use a hosted agent, while [editing your build definition](#), on the **Options** or **General** tab or **Process** step, for the **Agent queue**, select either:

- **Hosted VS2017** if your team uses Visual Studio 2017.
- **Hosted Linux** if your team uses development tools on Ubuntu.
- **Hosted macOS Preview** if your team uses development tools on macOS.

This option affects where your data is stored. [Learn more](#)

For manual selection of tool versions on this hosted agent, see **Q & A** below.

- **Hosted** if your team uses Visual Studio 2013 or Visual Studio 2015.

## Software

We update the software on the hosted agents once every month.

- [Inventory of software currently installed on the Hosted VS2017 agent](#).
- [Inventory of software currently installed on the Hosted Linux agent](#).
- [Inventory of software currently installed on the Hosted macOS Preview agent](#).
- [Inventory of software currently installed on the Hosted agent](#).

## Capabilities and limitations

Hosted agents:

- Run as a service.
- Have [the above software](#). You can also add software using [tool installers](#).
- Provide 10 GB of storage.

Hosted agents do not offer:

- The ability to log on.

- The ability to [drop artifacts to a UNC file share](#).
- The ability to run [XAML builds](#).
- Potential performance advantages that you might get by using private agents which might start and process builds faster. [Learn more](#)

If our hosted agents don't meet your needs, then you can [deploy your own private agents](#).

## Avoid hard coded references

When you use a hosted agent, you should always use [variables](#) to construct any references to resources used by your build. We recommend that you avoid making hard-coded presumptions about resources provided by the hosted agent (for example, the drive letter or folder that contains the repository).

## Q & A

### **I can't select a hosted agent and I can't queue my build or deployment. How do I fix this?**

By default, all project contributors in an account have access to the hosted agents. But, your account administrator may limit the access of hosted agents to select users or projects. Ask the owner of your VSTS account to grant you permission to use a hosted agent. See [agent queue security](#).

### **I need more agents. What can I do?**

A: All VSTS accounts are provided with a single agent and a limited number of free minutes each month. If you need more minutes, or need to run more than one build or release job concurrently, then you can [buy concurrent pipelines](#).

### **I'm looking for the hosted XAML build controller. Where did it go?**

The hosted XAML build controller is no longer supported. If you have an account in which you still need to run [XAML builds](#), you should set up a [private build server](#) and a [private build controller](#).

### **How can I manually select versions of tools on the Hosted macOS Preview agent?**

- **Xamarin**

To manually select a Xamarin SDK version to use on the **Hosted macOS Preview** agent, before your Xamarin build step, execute this command line as part of your build, replacing the Mono version number 5.4.1 as needed (also replacing '.' characters with underscores: '\_'). Choose the Mono version that is associated with the Xamarin SDK version that you need.

```
/bin/bash -c "sudo $AGENT_HOMEDIRECTORY/scripts/select-xamarin-sdk.sh 5_4_1"
```

Mono versions associated with Xamarin SDK versions on the **Hosted macOS Preview** agent can be found [here](#).

Note that this command does not select the Mono version beyond the Xamarin SDK. To manually select a Mono version, see instructions below.

- **Xcode**

If you use the [Xcode task](#) included with VSTS 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 Preview** agent, before your `xcodebuild` build step, execute this command line as part of your build, replacing the Xcode version number 8.3.3 as needed:

```
/bin/bash -c "sudo xcode-select -s /Applications/Xcode_8.3.3.app/Contents/Developer"
```

Xcode versions on the **Hosted macOS Preview** agent can be found [here](#).

- **Mono**

To manually select a Mono version to use on the **Hosted macOS Preview** agent, before your Mono build step, execute this script as part of your build, replacing the Mono version number 5.4.1 as needed:

```
SYMLINK=5_4_1
MONOPREFIX=/Library/Frameworks/Mono.framework/Versions/$SYMLINK
echo "##vso[task.setvariable
variable=DYLD_FALLBACK_LIBRARY_PATH;]$MONOPREFIX/lib:/lib:/usr/lib:$DYLD_LIBRARY_FALLBACK_PATH"
echo "##vso[task.setvariable
variable=PKG_CONFIG_PATH;]$MONOPREFIX/lib/pkgconfig:$MONOPREFIX/share/pkgconfig:$PKG_CONFIG_PATH"
echo "##vso[task.setvariable variable=PATH;]$MONOPREFIX/bin:$PATH"
```

**TIP**

We recommend that you [migrate your XAML builds to new builds](#).

# Deployment groups

2/26/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018

A deployment group is a logical set of deployment target machines that have agents installed on each one. Deployment groups represent the physical environments; for example, "Dev", "Test", "UAT", and "Production". In effect, a deployment group is just another grouping of agents, much like an [agent pool](#).

When authoring a VSTS or TFS Release definition, you can specify the deployment targets for a [phase](#) 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 **Build & Release** hub, 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 phase](#). You manage the security for a deployment group by [assigning security roles](#).

## Deploy agents to a deployment group

Every target machine in the deployment group requires the build and release agent to be installed. You can do this using the script that is generated in the **Deployment Groups** tab of the **Build & Release** hub. 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 **VSTS Agent** Azure VM extension on each of the VMs, or by using the **Azure Resource Group Deployment** task in your release definition to create a deployment group dynamically.

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 definition or a release, use the links in the **Releases** tab.

## Share a deployment group

Each deployment group is a member of a **deployment pool**, and you can share the deployment pool and groups across projects provided that:

- The user sharing the deployment pool has [User permission](#) for the pool containing the group.
- The user sharing the deployment pool has permission to create a deployment group in the project where it is being shared.
- The project does not already contain a deployment group that is a member of the same deployment pool.

The tags you assign to each machine in the pool are scoped at project level, so you can specify a different tag for the same machine in each deployment group.

### Add a deployment pool and group to another project

To manage a deployment pool, or to add an existing deployment pool and the groups it contains to another project, choose the **Manage** link in the **Agent Pool** section of the **Deployment Group** page. In the **Deployment Pools** page, select the projects for which you want the deployment group to be available, then save the changes.

When you navigate to the **Deployment Groups** page in the target project(s), you will see the deployment group you added and you can assign project-specific machine tags as required.

### Create a new deployment pool

You can add a new deployment pool to an account, share it amongst your projects, and then add deployment groups to it. In the **Deployment Pools** page, choose **+ New**. In the **New deployment pool** panel, enter a name for the pool and then select the projects for which you want it to be available.

When you navigate to the **Deployment Groups** page in the target project(s), you will see the deployment group you added and you can assign project-specific machine tags as required.

## Related topics

- [Run on machine group phase](#)
- [Deploy an agent on Windows](#)
- [Deploy an agent on macOS](#)
- [Deploy an agent on Linux](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Artifacts in Team Build

10/12/2017 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2015.3 and newer | TFS 2015 RTM (see Q&A)

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.

Release Management can pick up and use your build artifacts as part of a continuous integration (CI)/continuous deployment (CD) process. In this scenario, you're automatically building a web app with each commit using your CI build. Your CD release process picks up the .ZIP (ASP.NET or Node.js) or .WAR (Java) web deployment file. Your changes are automatically deployed to a test environment in Azure.

## Examples

Here are some examples of how to publish artifacts from the **Tasks** tab of your build definition.

### Publish a README.md file

#### TIP

If you want to try this and you don't already have a Git repo with a README.md file at the root, you can quickly [create one](#).



### Utility: Publish Build Artifacts

- Path to publish

```
$(Build.SourcesDirectory)/README.md
```

- Artifact name

```
drop
```

- Artifact publish location: Visual Studio Team Services/TFS (**TFS 2018 RTM and older**: Artifact type: Server)

### Two sets of artifacts

You can create multiple artifact items. For example:



### Utility: Publish Build Artifacts

- Path to publish

```
$(Build.SourcesDirectory)/README.md
```

- Artifact name

```
drop1
```

- Artifact publish location: Visual Studio Team Services/TFS (**TFS 2018 RTM and older**: Artifact type: Server)



### Utility: Publish Build Artifacts

- Path to publish

```
$(Build.SourcesDirectory)/README.md
```

- Artifact name

```
drop2
```

- Artifact publish location: Visual Studio Team Services/TFS (**TFS 2018 RTM and older**: Artifact type: Server)

The completed build delivers two sets of artifacts.

**Build succeeded**

Build 1693  
Ran for 14 seconds (Default), completed 7 seconds ago

Summary   Timeline   **Artifacts**   Code coverage\*   Tests

Name ↑		Download	Explore
drop1		Download	Explore
drop2		Download	Explore

You would probably never need to drop two copies of the same files. The point of this example is to show how you can drop multiple sets of artifacts that can be independently organized, explored, downloaded, and used by your deployment process.

### C++ app



### Utility: Copy Files

- Source folder

```
$(Build.ArtifactStagingDirectory)
```

- Contents

```
**/$(BuildConfiguration)/**/?(*.exe|*.dll|*.pdb)
```

- Target folder

```
$(Build.ArtifactStagingDirectory)
```



### Utility: Publish Build Artifacts

- Path to publish

```
$(Build.ArtifactStagingDirectory)
```

- Artifact name

```
drop
```

- Artifact publish location: Visual Studio Team Services/TFS (**TFS 2018 RTM and older**: Artifact type: Server)

## Tips

- Artifact publish location** argument: **Visual Studio Team Services/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 VSTS or TFS. But if you're using a private Windows agent, you've got the option [drop to a UNC file share](#).
- Artifact name** argument: Just enter a name that's meaningful to you.
- Use forward slashes in file path arguments so that they work for all agents. Backslashes don't work for macOS and Linux agents.
- On VSTS and some versions of TFS there are two different [variables](#) that point to the staging directory: `Build.ArtifactStagingDirectory` and `Build.StagingDirectory`. These are interchangeable.
- The directory referenced by `Build.ArtifactStagingDirectory` is cleaned up after each build.
- You can [get build artifacts from the REST API](#).

## Publish from TFS to UNC file share

If you're using a private Windows agent, you can set the **artifact publish location** option (**TFS 2018 RTM and older**: artifact type) to publish your files to a UNC **file share**.

### NOTE

Use a Windows build agent. This option doesn't work for macOS and Linux agents.

Choose file share to copy the artifact to a file share. Some common reasons to do this:

- The size of your drop is large and consumes too much time and bandwidth to copy.
- You need to run some custom scripts or other tools against the artifact.

If you use a file share, specify the UNC file path to the folder. You can control how the folder is created for each build using [variables](#). For example `\my\share\$(Build.DefinitionName)\$(Build.BuildNumber)`.

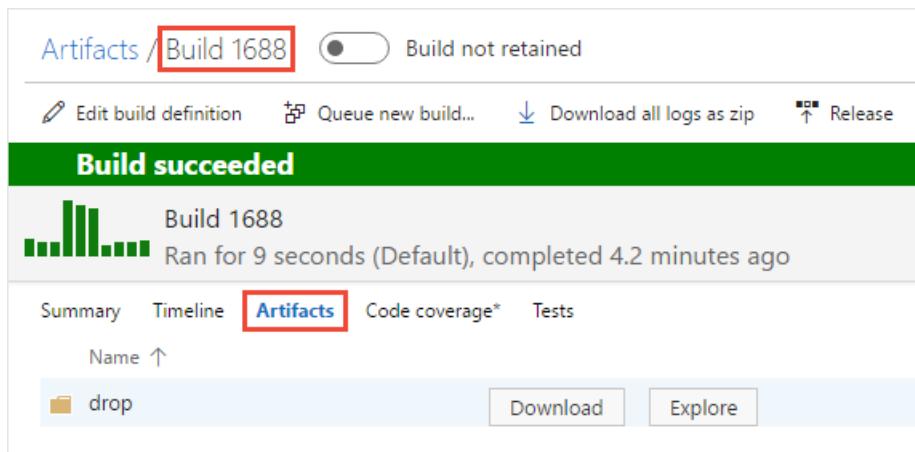
## Task reference

Use these tasks to publish artifacts:

-  [Utility: Copy Files](#) By copying files to `$(Build.ArtifactStagingDirectory)` you can publish multiple files of different types from different places specified by your [matching patterns](#).
-  [Utility: Delete Files](#) Handy to prune unnecessary files that you copied to the staging directory.
-  [Utility: Publish Build Artifacts](#)

## Explore, download, and deploy your artifacts

When the build is done, if you watched it run, click the name of the completed build and then click the artifacts tab to see your artifact.



The screenshot shows the build results for 'Build 1688'. The status bar at the top indicates 'Build succeeded'. Below it, there's a summary bar with a green bar chart icon, the text 'Build 1688', and 'Ran for 9 seconds (Default), completed 4.2 minutes ago'. Underneath, there are tabs for 'Summary', 'Timeline', 'Artifacts' (which is highlighted with a red box), 'Code coverage\*', and 'Tests'. A search bar labeled 'Name ↑' is present. Below the tabs, a list item 'drop' is shown with a 'Download' button and an 'Explore' button.

From here you can explore or download the artifacts.

You can also use Release Management to deploy your app using the artifacts that you've published. See [Artifacts in Release Management](#).

## Q&A

### How do I publish artifacts from TFS 2015?

If you're using TFS 2015 RTM, then the steps in the above examples are not available. Instead, you copy and publish your artifacts using a single task: [Build: Publish Build Artifacts](#).

# Build definition history

12/19/2017 • 1 min to read • [Edit Online](#)

## **VSTS | TFS 2018 | TFS 2017 | TFS 2015**

From the **History** tab you can see a list of changes that includes who made the change and when the change occurred.

**VSTS:** To work with a change, select it, click ..., and then click **Compare Difference** or **Revert Definition**.

**Team Foundation Server (TFS) 2017.1 and older:** After you've viewed the history, if you want details about a change, select it and then choose **Diff**. If you want to roll back to an earlier version, select it, and then click **Rollback**.

## Q&A

### **Can I edit the JSON source directly?**

No

### **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

### **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

### **I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build definition options

12/21/2017 • 6 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

Here we explain settings you can change on the build definition **Options** tab.

## Description

### Team Foundation Server (TFS) 2017.1 and older

This section is available under **General tab**.

If you specify a description here, it is shown near the name of the build definition when you select it in the Build area of your team project.

## Build number format

If you leave it blank, your completed build is given a unique integer as its name. But you can give completed builds much more useful names that are meaningful to your team. You can use a combination of tokens, variables, and underscore characters.

### Example

At the time the build is queued:

- Team project name: Fabrikam
- Build definition name: CIBuild
- Branch: master
- Build ID: 752
- Date: August 5, 2009.
- Time: 9:07:03 PM.
- The build ran once earlier today.

If you specify this build number format:

```
$(TeamProject)_$(BuildDefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)
```

Then the second completed build on this day would be named: **Fabrikam\_CIBuild\_master\_20090805.2**

### Tokens

The following table shows how each token is resolved based on the previous example.

TOKEN	EXAMPLE REPLACEMENT VALUE
-------	---------------------------

TOKEN	EXAMPLE REPLACEMENT VALUE
<code>\$(BuildDefinitionName)</code>	CIBuild  Note: The build definition name must not contain invalid or whitespace characters.
<code>\$(BuildID)</code>	752  \$(BuildID) is an internal immutable ID.
<code>\$(DayOfMonth)</code>	5
<code>\$(DayOfYear)</code>	217
<code>\$(Hours)</code>	21
<code>\$(Minutes)</code>	7
<code>\$(Month)</code>	8
<code>\$(Rev:.r)</code>	2 (The third build on this day will be 3, and so on.)  Use <b>\$(Rev:.rr)</b> to ensure that every completed build has a unique name. When a build is completed, if nothing else in the build number has changed, the Rev integer value is incremented by one.  If you want to show prefix zeros in the number, you can add additional 'r' characters. For example, specify <b>\$(rev:.rr)</b> if you want the Rev number to begin with 01, 02, and so on.
<code>\$(Date:yyyyMMdd)</code>	20090824  You can specify other date formats such as <b>\$(Date:MMddyy)</b>
<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 build number format. For example, if you've defined `My.Variable`, you could specify the following build number format:

```
$(Build.DefinitionName)_$(Build.DefinitionVersion)_$(Build.RequestedFor)_$(Build.BuildId)_$(My.Variable)
```

The first four variables are predefined. `My.Variable` is defined by you on the [variables tab](#).

## Badge enabled

### TFS 2017.1 and older

This section is available under **General tab**.

Select if you want to show the latest outcome of this build on external web sites.

1. Select the **Badge enabled** check box.
2. Save the definition.
3. When the **Show url** link appears, click it and copy the URL to your clipboard.
4. Use the URL as the source of an image in the HTML of the page of the external web site. For example:

```
.extraheader="AUTHORIZATION: basic *****" submodule
update --init -recursive
```

Use a secret variable in your project or build definition to store the personal access token (PAT) that you generate in VSTS or GitHub with access to your submodules. 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 is not desirable in automated builds.

## Checkout files from LFS

Select if you want to download files from [large file storage \(LFS\)](#).

- **VSTS, TFS 2017.3 and newer:** Select the check box to enable this option.
- **TFS 2017 RTM and TFS 2015 (macOS and Linux only):** On the **Variables** tab, set `Agent.Source.Git.Lfs` to `true`.

If you're using TFS, or if you're using VSTS with a private agent, then you must install git-lfs on the agent to make this option work.

## Don't sync sources (TFS 2017 and newer only)

Use this option if you want to skip fetching new commits. This option can be useful in cases such as when you want to:

- Git init, config, and fetch using your own custom options.
- Use a build process to just run automation (for example some scripts) that do not depend on code in version control.

If you want to disable downloading sources:

- **VSTS, 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 process more efficient. Your repository might be large if it has been in use for a long time. 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.

### VSTS, 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.

# TFVC options

FEATURE	VSTS, TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Clean	Yes	Yes
Specify local path	Yes	No
Label sources	Yes	No

## NOTE

**VSTS, 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 process requires. If a subfolder of a mapped folder contains files that the build process does not require, map it with a type value of **Cloak**.

Make sure that you **Map** all folders that contain files that your build process 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 team project is mapped in the workspace. This configuration results in the build agent downloading all the files in the version control folder of your team project. If this folder contains lots of data, your build could waste build system resources and slow down your build process 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 kinds of cleaning of the working directory of your private agent before the build is run.

## NOTE

Cleaning is not relevant if you are using a [hosted agent](#) because you get a new agent every time in that case.

In general, for faster performance by your private agents, don't clean the repo. In this case to get the most performance advantage, make sure you are also building incrementally. For example, if you are building Visual Studio projects, make sure to clear the **Clean** check box of the Visual Studio Build or MSBuild task.

If you do need to clean the repo (for example to avoid problems caused by residual files from a previous build), below are the options you've got.

### VSTS, 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 process 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 process 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 `$(Agent.BuildDirectory)`, which is the entire working folder that contains the sources folder, binaries folder, artifact folder, and so on.
- `source` if you want to delete `$(Build.SourcesDirectory)` .
- `binary` If you want to delete `$(Build.BinariesDirectory)` .

#### TFS 2015 RTM

Select **true** to delete the repository folder.

### Label sources

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

#### NOTE

You can only use this feature when the source repository in your build is a Git or TFVC repository from your team 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` is defined by you on the [variables tab](#).

The build process labels your sources with a [TFVC label](#).

## Q&A

### How do I reference the directories on the build agent?

Reference directories using build variables such as `$(Build.SourcesDirectory)` and `$(Build.BinariesDirectory)` . To learn more, see [Build variables](#).

### What protocols can the build agent use with Git?

We support HTTPS.

We don't yet support SSH. See [User Voice: Allow build to use ssh authentication while checking out git sub modules](#)

### What is scorch?

Schorch is a TFVC power tool. See [Microsoft Visual Studio Team Foundation Server 2015 Power Tools](#) .

### **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

### **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

### **I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

### **Where is the guidance for TFS 2013?**

[Work with build workspaces \(XAML builds\)](#)

# Build definition triggers

12/19/2017 • 5 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)**

On the **Triggers** tab you specify the events that will trigger the build. You can use the same build definition for both CI and scheduled builds.

## Continuous integration (CI)

Select this trigger if you want the build to run whenever someone checks in code.

### Batch changes

Select this check box if you have a lot of 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.

If you are using batched changes, you can also specify a maximum number of concurrent builds per branch.

You can batch changes when your code is in Git in the team project or on GitHub. This option is not available if your code is in a remote Git repo or in Subversion.

### Git filters

If your repository is Git then you can specify the branches where you want to trigger builds. You can use wildcard characters.

#### Path filters in VSTS and Team Foundation Services (TFS)

If your Git repo is in VSTS or TFS, you can also specify path filters to reduce the set of files that you want to trigger a build.

#### Tips:

- If you don't set path filters, then the root folder of the repo is implicitly included by default.
- When you add an explicit path filter, the implicit include of the root folder is removed. So make sure to explicitly include all folders that your build needs.
- 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.

#### Example

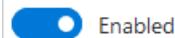
For example, you want your build to be triggered by changes in master and most, but not all, of your feature branches. You also don't want builds to be triggered by changes to files in the tools folder.

#### VSTS

## Continuous Integration

Build every change to matching branches

Disable this trigger



Enabled

### Repositories



FabrikamFibre ^

Build when any branch changes

Batch changes while a build is in progress

#### Branch Filters

Type Branch specification

Include	▼	master	▼	
Include	▼	features/*	▼	
Exclude	▼	features/not-this-one	▼	

Add

#### Path Filters

Type Path specification

Exclude	▼	/tools	
Include	▼	/	

Add

## TFS 2017.1 and older versions

**Continuous integration (CI)**

Build each check-in.

Batch changes

#### Branch filters

	Include	▼	master	▼	
	Include	▼	features/*	▼	
	Exclude	▼	features/not-this-one	▼	

Add new filter

#### Path filters

	Exclude	▼	/tools	
	Include	▼	/	

Add new filter

## TFVC Include

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 on the [Repository tab](#).

## CI trigger for a remote Git repo or Subversion

You can also select the CI trigger if your code is in a remote Git repo or Subversion. In this case we poll for changes at a regular interval. For this to work, VSTS or your Team Foundation Server must be able to resolve the network address of the service or server where your code is stored. For example if there's a firewall blocking the connection, then the CI trigger won't work.

## Scheduled

Select the days and times when you want to run the build.

If your repository is Git, GitHub, or External Git, then you can also specify branches to include and exclude. You can use wildcards.

### Example: Nightly build of Git repo in multiple time zones

#### VSTS

The screenshot shows the VSTS build scheduling interface with two parallel schedules defined:

- Schedule 1 (Top):** Mon through Fri at 3:00 UTC+05:30 (Chennai, Kolkata, Mumbai, New Delhi).
  - When to build:** Mon, Tue, Wed, Thu, Fri (selected), Sat, Sun (disabled).
  - Time:** 03h, 00m, (UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi.
- Schedule 2 (Bottom):** Mon through Fri at 3:00 UTC-05:00 (Eastern Time (US & Canada)).
  - When to build:** Mon, Tue, Wed, Thu, Fri (selected), Sat, Sun (disabled).
  - Time:** 03h, 00m, (UTC-05:00) Eastern Time (US & Canada).

**Branch Filters:**

- Schedule 1:** Type: Include, Branch specification: features/india/\*
- Schedule 2:** Type: Include, Branch specification: features/nc/\*

**Additions:** + Add, + Add new time.

#### TFS 2017.1 and older versions

**Scheduled**

Build matching branches for each schedule.

[+ Add new time...](#)

24h time: 03 : 00 (UTC) Coordinated Universal Time [▼](#)

Su  M  Tu  W  Th  F  Sa

Branches

[X](#) [Include](#) [▼](#) [89](#) features/nc/\* [?](#)

[+ Add new branch](#)

---

24h time: 03 : 00 (UTC+05:30) Chennai, Kolkata, Mumbai, New Delhi [▼](#)

Su  M  Tu  W  Th  F  Sa

Branches

[X](#) [Include](#) [▼](#) [89](#) features/india/\* [?](#)

[+ Add new branch](#)

### Example: Nightly build with different frequencies

#### VSTS

**Scheduled**  
Build matching branches for each schedule

Disable this trigger [Enabled](#)

Schedules

[Mon through Fri at 3:00](#) [^](#) [Delete](#)

When to build

<input checked="" type="checkbox"/> Mon	<input checked="" type="checkbox"/> Tue	<input checked="" type="checkbox"/> Wed	<input checked="" type="checkbox"/> Thu	<input checked="" type="checkbox"/> Fri	Sat	Sun
03h <a href="#">▼</a>	00m <a href="#">▼</a>	(UTC) Coordinated Universal Time <a href="#">▼</a>				

Branch Filters

Type	Branch specification
Include	<a href="#">▼</a> master <a href="#">Delete</a>
Include	<a href="#">▼</a> releases/* <a href="#">Delete</a>
Include	<a href="#">▼</a> releases/lastversion <a href="#">Delete</a>

[+ Add](#)

[Sun at 3:00](#) [^](#) [Delete](#)

When to build

Mon	Tue	Wed	Thu	Fri	Sat	<input checked="" type="checkbox"/> Sun
03h <a href="#">▼</a>	00m <a href="#">▼</a>	(UTC) Coordinated Universal Time <a href="#">▼</a>				

Branch Filters

Type	Branch specification
Include	<a href="#">▼</a> releases/lastversion <a href="#">Delete</a>

[+ Add](#)

#### TFS 2017.1 and older versions

**Scheduled**  
Build matching branches for each schedule.

**Add new time...**

24h time: 03  : 00  (UTC) Coordinated Universal Time

Su  M  Tu  W  Th  F  Sa

Branches

- Include   master
- Include   releases/\*
- Exclude   releases/lastversion

**Add new branch**

24h time: 03  : 00  (UTC) Coordinated Universal Time

Su  M  Tu  W  Th  F  Sa

Branches

- Include   releases/lastversion

**Add new branch**

## TFVC gated check-in

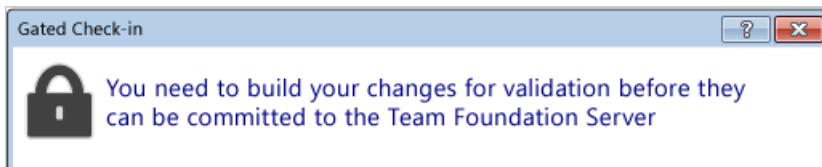
If your code is in a [Team Foundation version control \(TFVC\)](#) repo, use gated check-in to protect against breaking changes.

By default **Use workspace mappings for filters** is selected. Builds are triggered whenever a change is checked in under a path specified in your mappings in the [source repository settings](#).

Otherwise, you can clear this check box and specify the paths in the trigger.

### How it affects your developers

When a developer tries to check-in, they are prompted to build their changes.



The system then creates a shelveset and builds it.

For details on the gated check-in experience, see [Check in to a folder that is controlled by a gated check-in build process](#).

### 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 process does not add **\*\*\*NO\_CI\*\*\*** to the changeset description. As a result, CI builds that are affected by the check-in are run.

### A few other things to know

- Make sure the folders you include in your trigger are also included in your mappings on the [Repository tab](#).
- You can run gated builds on either a [hosted agent](#) or a [private agent](#).

# Q&A

## How do I protect my Git codebase from build breaks?

If your code is in a Git repo on VSTS or Team Foundation Server, you can create a branch policy that runs your build. See [Improve code quality with branch policies](#). This option is not available for GitHub repos.

## My build didn't run. What happened?

If your build is in VSTS, then at least one of your users must sign in regularly for CI and scheduled builds to run. Your VSTS account goes dormant five minutes after the last user signed out. After that, each of your build definitions will run one more time.

For example, while your account is dormant:

- A nightly build of code in your VSTS account will run only one night until someone signs in again.
- CI builds of an external Git repo will stop running until someone signs in again.

## Can I chain builds so that one build triggers another?

Not yet. See [User Voice: Provide build configuration dependencies in TFS Build](#).

## Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

## I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

## I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build variables

1/18/2018 • 11 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)**

Variables give you a convenient way to get key bits of data into various parts of your build process.

USE	USER-DEFINED	PREDEFINED, ALL SCOPES	PREDEFINED, AGENT SCOPE	FORMAT	EXAMPLES AND MORE INFORMATION
As arguments to build steps	Yes	Yes	Yes	<code>\$(Build.DefinitionName)</code>	<a href="#">Command line</a> , <a href="#">Copy files</a>
Apply a version control label during the build process	Yes	Yes	No	<code>\$(Build.DefinitionName)</code>	<a href="#">Repository tab</a> (Git and Team Foundation version control) <a href="#">Label format</a>
Customize the build number	Yes	Yes	No	<code>\$(Build.DefinitionName)</code>	<a href="#">Build number format options</a>
Environment variable in Windows batch scripts	Yes	Yes	Yes	<code>%BUILD_DEFINITIONNAME%</code>	<a href="#">PowerShell script</a>
Environment variable in PowerShell scripts	Yes	Yes	Yes	<code>\$env:BUILD_DEFINITIONNAME</code>	<a href="#">PowerShell script</a>
Environment variable in Shell scripts	Yes	Yes	Yes	<code>\$BUILD_DEFINITIONNAME</code>	<a href="#">Shell script</a>

## User-defined variables

Variables are a great way to store and share key bits of data in your build definition. Some build templates automatically define some variables for you.

For example, when you [create a new .NET app build](#), `BuildConfiguration` and `BuildPlatform` are automatically defined for you.

User-defined variables are formatted differently in different contexts. See above table.

### Secret Variables

We recommend that you make the variable  **Secret** if it contains a password, keys, or some other kind of data that you need to avoid exposing.

Tasks	Variables	Triggers	Options	Retention	History
Process variables					
	Name		Value		Settable at queue time
	system.collectionId		189a025d-3690-49db-91e8-e9e320834f58		
	system.teamProject		FabrikamFibre		
	system.definitionId		31		
	system.debug		false	<input checked="" type="checkbox"/>	
	BuildConfiguration		release	<input checked="" type="checkbox"/>	
	BuildPlatform		any cpu	<input checked="" type="checkbox"/>	
	password		*****		<input type="checkbox"/>
	<a href="#">+ Add</a>				

## Team Foundation Server (TFS) 2017.1 and older

Definitions / OurBuild | Builds

Build	Options	Repository	Variables	Triggers	General	Retention	History
Save	Queue build...	Undo					
List of predefined variables							
Name	Value					Allow at Queue Time	
system.collectionId	d72ef65b-fbd0-4ce1-bb0d-eb6393afc3b1					<input type="checkbox"/>	
system.teamProject	Scripts					<input type="checkbox"/>	
system.definitionId	98					<input type="checkbox"/>	
X system.debug	false					<input checked="" type="checkbox"/>	
X password	*****					<input type="checkbox"/>	
+ Add variable							

Secret variables are:

- Encrypted at rest with a 2048-bit RSA key.
- Not returned back to the client. They are automatically masked out of any log output from the build or release.
- Not decrypted into environment variables. So scripts and programs run by your build steps are not given access by default.
- Decrypted for access by your build steps. So you can use them in password arguments and also pass them explicitly into a script or a program from your build step (for example as `$(password)` ).

### Allow at queue time

Select this check box if you want to enable your team to modify the value when they manually queue a build.

### Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command.

**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 `secretSauce` variables

```
@echo ##vso[task.setvariable variable=sauce]crushed tomatoes
@echo ##vso[task.setvariable variable=secretSauce;issecret=true]crushed tomatoes with garlic
```



Read the variables

### Arguments

```
"$(sauce)" "$(secretSauce)"
```

### Script

```
@echo off
set sauceArgument=%~1
set secretSauceArgument=%~2
@echo No problem reading %sauceArgument% or %SAUCE%
@echo But I cannot read %SECRETSAUCE%
@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)
```

## Control variables

VARIABLE NAME	DESCRIPTION
Build.Clean	Modifies how the build agent cleans things up. See <a href="#">Source repositories</a> .
System.Debug	If you need more detailed logs to debug build problems, define and set it to <code>true</code> .

## Environment variables

You can pass environment variables of the build machine into build steps. For example, on the [Build tab](#) of a build definition, add this step:

TASK	ARGUMENTS
 Utility: Command Line	Tool: echo Arguments: \$(PATH)

#### NOTE

If you have defined the a variable of the same name (for example `PATH`) on the [variables tab](#), then your value overrides the environment variable when you use it as shown above.

## Predefined variables

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
Agent.BuildDirectory AGENT_BUILDDIRECTORY	Agent	The local path on the agent where all folders for a given build definition are created. For example: <code>c:\agent\_work\1</code>
Agent.HomeDirectory AGENT_HOME DIRECTORY	Agent	The directory the agent is installed into. This contains the agent software. For example: <code>c:\agent</code> . If you are using an on-premises agent, this directory is specified by you. See <a href="#">Agents</a> .
Agent.Id AGENT_ID	Agent	The ID of the agent.
Agent.JobStatus AGENT_JOBSTATUS	Agent	The status of the build. <ul style="list-style-type: none"><li>• <code>Canceled</code></li><li>• <code>Failed</code></li><li>• <code>Succeeded</code></li><li>• <code>SucceededWithIssues</code> (partially successful)</li></ul>
Agent.MachineName AGENT_MACHINENAME	Agent	The name of the machine on which the agent is installed.
Agent.Name AGENT_NAME	Agent	The name of the agent that is registered with the pool. If you are using an on-premises agent, this directory is specified by you. See <a href="#">agents(../concepts/agents/agents.md)</a> .

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
Agent.WorkFolder AGENT_WORKFOLDER	Agent	The working directory for this agent. For example: <code>c:\agent\_work</code> .
Build.ArtifactStagingDirectory BUILD_ARTIFACTSTAGINGDIRECTORY	Agent	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination.</p> <p>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 <a href="#">Copy files</a> and <a href="#">Publish build artifacts</a> steps.</p> <p><b>Note:</b> This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable.</p> <p><a href="#">See Artifacts in Team Build</a></p>
Build.BuildId BUILD_BUILDID	All	The ID of the record for the completed build.
Build.BuildNumber BUILD_BUILDPART	Agent, label format (see Notes)	<p>The name of the completed build.</p> <p>You can specify the build number format that generates this value in the <a href="#">definition options</a>.</p> <p>A typical use of this variable is to make it part of the label format, which you specify on the <a href="#">repository tab</a>.</p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the <a href="#">label format</a> will fail.</p>
Build.BuildUri BUILD_BUILDURI	Agent	The URI for the build. For example: <code>vstfs:///Build/Build/1430</code> .
Build.BinariesDirectory BUILD_BINARIESDIRECTORY	Agent	<p>The local path on the agent you can use as an output folder for compiled binaries. For example: <code>c:\agent\_work\1\b</code>.</p> <p>By default, new build definitions are not set up to clean this directory. You can define your build to clean it up on the <a href="#">Repository tab</a>.</p>

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
Build.DefinitionName  BUILD_DEFINITIONNAME	All (see Notes)	The name of the build definition.  Note: This value can contain whitespace or other invalid label characters. In these cases, the <a href="#">label format</a> will fail.
Build.DefinitionVersion  BUILD_DEFINITIONVERSION	All	The version of the build definition.
Build.QueuedBy  BUILD_QUEUEDEDBY	All (see Notes)	<a href="#">How are the identity variables set?</a>  Note: This value can contain whitespace or other invalid label characters. In these cases, the <a href="#">label format</a> will fail.
Build.QueuedById  BUILD_QUEUEDEDBYID	All	<a href="#">How are the identity variables set?</a>
Build.Reason  BUILD_REASON	All	<p><b>VSTS Only</b></p> <p>The event that caused the build to run.</p> <ul style="list-style-type: none"> <li>• <code>Manual</code> : A user manually queued the build.</li> <li>• <code>IndividualCI</code> : <b>Continuous integration (CI)</b> triggered by a Git push or a TFVC check-in.</li> <li>• <code>BatchedCI</code> : <b>Continuous integration (CI)</b> triggered by a Git push or a TFVC check-in, and the <b>Batch changes</b> was selected.</li> <li>• <code>Schedule</code> : <b>Scheduled</b> trigger.</li> <li>• <code>validateShelveset</code> : A user manually queued the build of a specific TFVC shelveset.</li> <li>• <code>CheckInShelveset</code> : <b>Gated check-in</b> trigger.</li> <li>• <code>PullRequest</code> : The build was triggered by a Git branch policy that requires a build.</li> </ul> <p>See <a href="#">Build definition triggers</a>, <a href="#">Improve code quality with branch policies</a>.</p>
Build.Repository.Clean  BUILD_REPOSITORY_CLEAN	Agent	The value you've selected for <b>Clean</b> in the <a href="#">source repository settings</a> .

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
Build.Repository.LocalPath BUILD_REPOSITORY_LOCALPATH	Agent	<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 definitions update only the changed files. You can modify how files are downloaded on the <a href="#">Repository tab</a>.</p>
Build.Repository.Name BUILD_REPOSITORY_NAME	Agent	The name of the <a href="#">repository</a> .
Build.Repository.Provider BUILD_REPOSITORY_PROVIDER	Agent	<p>The type of <a href="#">repository</a> you selected.</p> <ul style="list-style-type: none"> <li><code>TfsGit</code> : <a href="#">TFS Git repository</a></li> <li><code>TfsVersionControl</code> : <a href="#">Team Foundation Version Control</a></li> <li><code>Git</code> : Git repository hosted on an external server</li> <li><code>GitHub</code></li> <li><code>Svn</code> : Subversion</li> </ul>
Build.Repository.Tfvc.Workspace BUILD_REPOSITORY_TFVC_WORKSPACE	Agent	<p>Defined if your <a href="#">repository</a> is Team Foundation Version Control. The name of the <a href="#">TFVC workspace</a> 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>
Build.Repository.Uri BUILD_REPOSITORY_URI	Agent	<p>The URL for the repository. For example:</p> <ul style="list-style-type: none"> <li>Git:  <code>https://fabrikamfiber.visualstudio.com/_git/Scripts</code></li> <li>TFVC:  <code>https://fabrikamfiber.visualstudio.com/</code></li> </ul>
Build.RequestedFor BUILD_REQUESTEDFOR	All (see Notes)	<p><a href="#">How are the identity variables set?</a></p> <p>Note: This value can contain whitespace or other invalid label characters. In these cases, the <a href="#">label format</a> will fail.</p>
Build.RequestedForEmail BUILD_REQUESTEDFOREMAIL	All	<a href="#">How are the identity variables set?</a>

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
Build.RequestedForId BUILD_REQUESTEDFORID	All	<p><a href="#">How are the identity variables set?</a></p>
Build.SourceBranch BUILD_SOURCEBRANCH	All (see Notes)	<p>The branch the build was queued for. Some examples:</p> <ul style="list-style-type: none"> <li>• Git repo branch: <code>refs/heads/master</code></li> <li>• Git repo pull request: <code>refs/pull/1/merge</code></li> <li>• TFVC repo branch: <code>\$/teamproject/main</code></li> <li>• TFVC repo gated check-in: <code>Gated_2016-06-06_05.20.51.4369;username@live.com</code></li> <li>• TFVC repo shelveset build: <code>myshelveset;username@live.com</code></li> </ul> <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 BUILD_SOURCEBRANCHNAME	All (see Notes)	<p>The name of the branch the build was queued for.</p> <ul style="list-style-type: none"> <li>• 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>.</li> <li>• 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>.</li> <li>• 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>.</li> </ul> <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>

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
Build.SourcesDirectory BUILD_SOURCESDIRECTORY	Agent	<p>The local path on the agent where your source code files are downloaded. For example:  <span style="border: 1px solid black; padding: 2px;">c:\agent\_work\1\s</span></p> <p>By default, new build definitions update only the changed files. You can modify how files are downloaded on the <a href="#">Repository tab</a>.</p>
Build.SourceVersion BUILD_SOURCEVERSION	Agent	<p>The latest version control change that is included in this build.</p> <ul style="list-style-type: none"> <li>• Git: The <a href="#">commit ID</a>.</li> <li>• TFVC: the <a href="#">changeset</a>.</li> </ul>
Build.StagingDirectory BUILD_STAGINGDIRECTORY	Agent	<p>The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: <span style="border: 1px solid black; padding: 2px;">c:\agent\_work\1\a</span></p> <p>A typical way to use this folder is to publish your build artifacts with the <a href="#">Copy files</a> and <a href="#">Publish build artifacts</a> steps.</p> <p><b>Note:</b> This directory is purged before each new build, so you don't have to clean it up yourself.</p> <p>Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable.</p> <p>See <a href="#">Artifacts in Team Build</a></p>
Build.Repository.Git.SubmoduleCheckout BUILD_REPOSITORY_GIT_SUBMODULE_CHECKOUT	Agent	<p>The value you've selected for <b>Checkout submodules</b> on the <a href="#">repository tab</a>.</p>
Build.SourceTfvcShelveset BUILD_SOURCETFVCSHELVESET	All (see Notes)	<p>Defined if your <a href="#">repository</a> is Team Foundation Version Control.</p> <p>If you are running a <a href="#">gated build</a> or a <a href="#">shelveset build</a>, this is set to the name of the <a href="#">shelveset</a> 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 COMMON_TESTRESULTSDIRECTORY	Agent	<p>The local path on the agent where the test results are created. For example: <span style="border: 1px solid black; padding: 2px;">c:\agent\_work\1\TestResults</span></p>

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
System.AccessToken SYSTEM_ACCESTOKEN	Agent	Use the OAuth token to access the REST API.
System.CollectionId SYSTEM_COLLECTIONID	All	The GUID of the team foundation collection.
System.DefaultWorkingDirectory SYSTEM_DEFAULTWORKINGDIRECTORY	Agent	<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 definitions update only the changed files. You can modify how files are downloaded on the <a href="#">Repository tab</a>.</p>
System.DefinitionId SYSTEM_DEFINITIONID	All	The ID of the build definition.
System.PullRequest.IsFork SYSTEM_PULLREQUEST_ISFORK	All	If the pull request is from a fork of the repository, this variable is set to <code>True</code> . Otherwise, it is set to <code>False</code> .
System.PullRequest.PullRequestId SYSTEM_PULLREQUEST_PULLREQUESTID	All	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 <a href="#">Git PR affected by a branch policy</a> .)
System.PullRequest.SourceBranch SYSTEM_PULLREQUEST_SOURCEBRANCH	All	<p>The branch that is being reviewed in a pull request. For example:  <code>refs/heads/users/raisa/new-feature</code></p> <p>(This variable is initialized only if the build ran because of a <a href="#">Git PR affected by a branch policy</a>.)</p>
System.PullRequest.SourceRepositoryURI SYSTEM_PULLREQUEST_SOURCEREPOURI	All	<p><b>VSTS Only</b></p> <p>The URL to the repo that contains the pull request. For example:  <code>https://ouraccount.visualstudio.com/_git/OurProject</code></p> <p>(This variable is initialized only if the build ran because of a <a href="#">VSTS Git PR affected by a branch policy</a>. It is not initialized for GitHub PRs.)</p>

VARIABLE NAME ENVIRONMENT VARIABLE NAME	SCOPE	NOTES
System.PullRequest.TargetBranch SYSTEM_PULLREQUEST_TARGETBRA NCH	All	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 <a href="#">Git PR affected by a branch policy</a> .)
System.TeamFoundationCollectionUri SYSTEM_TEAMFOUNDATIONCOLLEC TIONURI	Agent	The URI of the team foundation collection. For example: <code>https://fabrikamfiber.visualstudio.com/</code> .
System.TeamProject SYSTEM_TEAMPROJECT	All	The name of the team project that contains this build.
System.TeamProjectId SYSTEM_TEAMPROJECTID	All	The ID of the team project that this build belongs to.
TF_BUILD	Agent	Set to <code>True</code> if the script is being run by a build step.

## Q&A

### What are the predefined Release Management variables?

#### Default release management variables

### 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 <a href="#">Continuous integration (CI) triggers</a>	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 <a href="#">branch policy build</a> .	The system identity, for example: <code>[DefaultCollection]\Project Collection Service Accounts</code>	The person who checked in the changes.
In TFVC by a <a href="#">gated check-in trigger</a>	The person who checked in the changes.	The person who checked in the changes.
In Git or TFVC by the <a href="#">Scheduled triggers</a>	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 <b>Queue build</b> button	You	You

### **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

### **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

### **I use TFS on-premises and I don't see some of these features. Why not?**

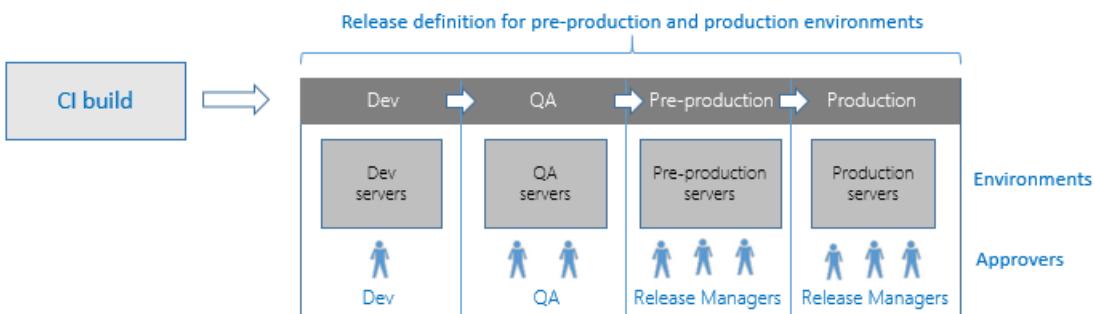
Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# What is Release Management?

2/28/2018 • 4 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017 | TFS 2015

**Release Management** is a service in Visual Studio Team Services (VSTS) and Team Foundation Server (TFS 2015.2 and later) and an essential element of DevOps that helps your team **continuously deliver** software to your customers at a faster pace and with lower risk. You can **fully automate** the testing and delivery of your software in multiple environments all the way to production, or set up semi-automated processes with **approvals** and **on-demand deployments**.



1. **Watch this video** - see Release Management in action.

<https://www.youtube.com/embed/zSPuRXTeZW8>

2. **Decide if it suits your scenarios** - use the simple checklist.
3. **See how it works** - get a basic understanding of the process.
4. **Get started now** - follow the steps to deploy your apps.

## Is Release Management for you?

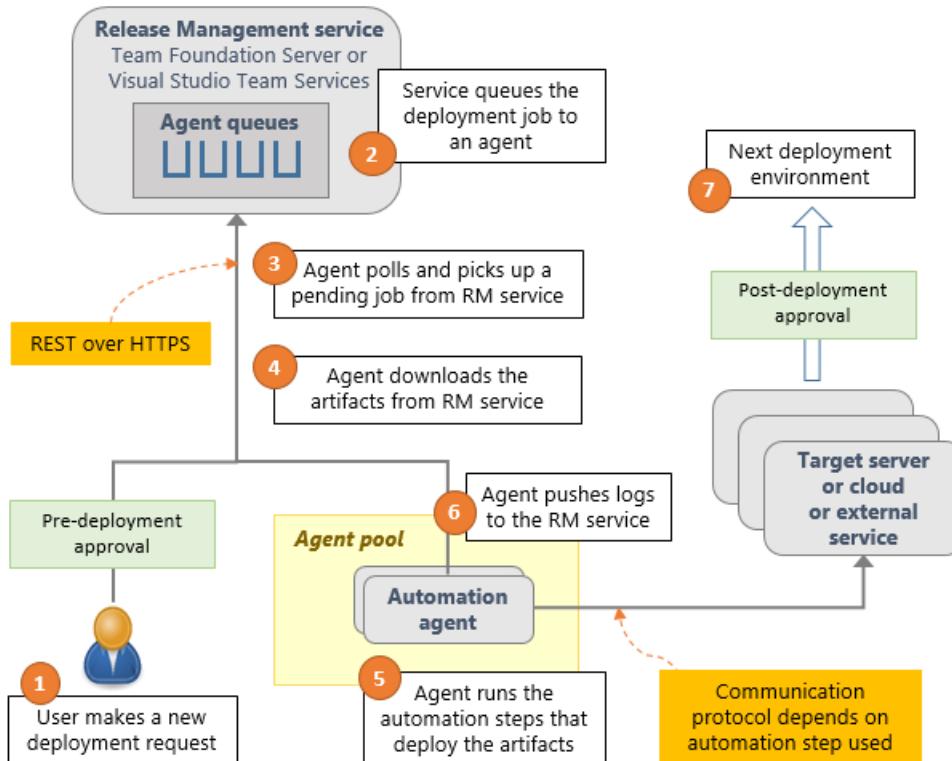
Consider using Release Management if:

- **You develop applications and need to deploy them regularly to any platform**, public or private cloud services, or App stores. Release Management has many out-of-the-box tasks to deploy a variety of applications. If you cannot find an out-of-the-box task to deploy your application using Release Management, consider this: if you can script the deployment of your application using Shell scripts or PowerShell scripts, utilities such as Ant or Maven, batch files or EXE utilities, then you can deploy it using Release Management. Release Management also integrates with third party deployment systems such as Chef and Docker.
- **You use a continuous integration (CI) system** and are looking for a fully-fledged continuous delivery or release management system. Whether you use Team Build from VSTS or TFS, or Jenkins as your CI system, you can set up Release Management to automatically deploy new builds to multiple environments. Even if we do not yet support integration with your favorite CI system or artifact repository, you can still write custom tasks to download and deploy artifacts from it.
- **You need to track the progress of releases**. If you use several environments for your tests, Release Management helps you monitor whether a release has been deployed and tested on each of these environments. Release Management also tracks whether an issue fixed by a developer, or a product backlog item completed by your team, has been deployed to a specific environment.

- **You need control of the deployments.** Release Management lets you specify which users can change the configuration of an environment, or approve the release to be deployed into a particular environment. If there is a problem with your deployment, Release Management helps you roll back to a previous deployment, and provide all the logs in one place to help you debug the problem.
- **You need audit history for all releases and their deployments.** Release Management provides a history of all changes to the definitions, configurations, and deployments. It also provides a history of all the activity performed during each deployment. Each release is accompanied by a listing of new features and developer commits that went into that release.

## How does Release Management work?

The Release Management service stores the data about your release definitions, environments, tasks, releases, and deployments in VSTS or TFS.



Release Management runs the following steps as part of every deployment:

1. **Pre-deployment approval:** When a new deployment request is triggered, Release Management checks whether a pre-deployment approval is required before deploying a release to an environment. If it is required, it sends out email notifications to the appropriate approvers.
2. **Queue deployment job:** Release Management schedules the deployment job on an available [automation agent](#). An agent is a piece of software that is capable of running tasks in the deployment.
3. **Agent selection:** An automation agent picks up the job. The agents for Release Management are exactly the same as those that run your Builds in VSTS and TFS. A release definition can contain settings to select an appropriate agent at runtime.
4. **Download artifacts:** The agent then runs all the tasks in the deployment job to deploy the app to the target servers for an environment.
5. **Run the deployment tasks:** The agent runs the automation steps that deploy the artifacts to the target server (6).
6. **Push logs:** The agent pushes logs to the RM service (7).
7. **Post-deployment approval:** The RM service sends a REST over HTTPS response back to the user (8).
8. **Next deployment environment:** The process repeats for the next deployment environment.

6. **Generate progress logs:** The agent creates detailed logs for each step while running the deployment, and pushes these logs back to VSTS or TFS.
7. **Post-deployment approval:** When deployment to an environment is complete, Release Management checks if there is a post-deployment approval required for that environment. If no approval is required, or upon completion of a required approval, Release Management proceeds to trigger deployment to the next environment.

## Get started now!

Simply follow these steps:

1. [Deploy your Azure Web App quickly and simply](#)
2. [Set up a multi-stage managed release pipeline](#)
3. [Manage deployments by using approvals and gates](#)

## Related topics

- [Download Team Foundation Server](#)
- [Install and configure Team Foundation Server](#)
- [Sign up for VSTS](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Release definitions in Release Management

2/28/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

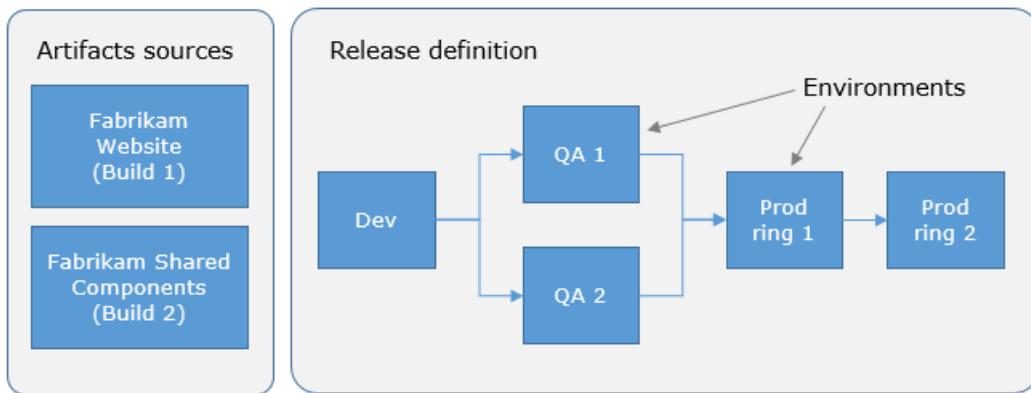
A **release definition** is one of the fundamental concepts in Release Management for VSTS and TFS. It defines the end-to-end release process for an application to be deployed across various environments.

You start using Release Management by authoring a release definition for your application. To author a release definition, you must specify the **artifacts** that make up the application and the **release process**.

An **artifact** is a deployable component of your application. It is typically produced through a Continuous Integration or a build process. Release Management can deploy artifacts that are produced by a [wide range of artifact sources](#) such as Team Build, Jenkins, or Team City.

You define the **release process** using [environments](#), and restrict deployments into or out of an environment using [approvals](#). You define the automation in each environment using [phases](#) and [tasks](#). You use [variables](#) to generalize your automation and [triggers](#) to control when the deployments should be kicked off automatically.

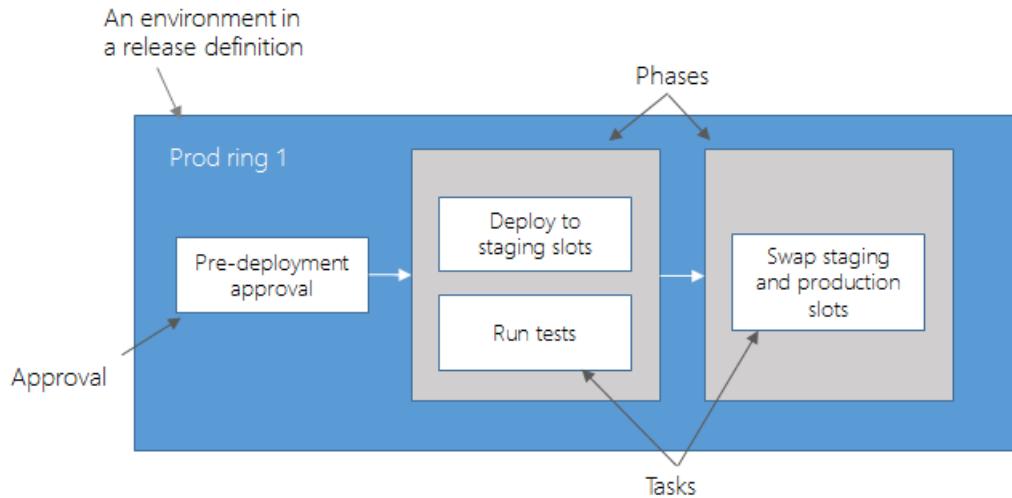
An example of a release process that can be modeled through a release definition is shown below:



## What's the difference between a release definition and a release?

In this example, a release of a website is created by collecting specific versions of two builds (artifacts), each from a different build definition. The release is first deployed to a Dev environment and then forked in parallel to two QA environments in parallel. If the deployment succeeds in both the QA environments, the release is deployed to Prod ring 1 and then to Prod ring 2. Each production ring represents multiple instances of the same website deployed at various locations around the globe.

An example of how deployment automation can be modeled within an environment is shown below:



In this example, a **phase** is used to deploy the web and database tiers to websites across the globe in parallel within production ring 1. Once all of those deployments are successful, a second phase is used to switch traffic from the previous version to the newer version.

**TFS 2015:** Phases, and fork and join deployments, are not available in TFS 2015.

Besides the release process, release definitions have a few options that can be customized: [release names](#) and [retention policies](#).

## Release names

The names of releases for a release definition 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 definition, edit the **Release name format** property.

When specifying the format mask, you can use the following pre-defined variables.

VARIABLE	DESCRIPTION
<b>Rev:r</b>	An auto-incremented number with at least the specified number of digits.
<b>Date / Date:MMddyy</b>	The current date, with the default format <b>MMddyy</b> . Any combinations of M/MM/MMM/MMMM, d/dd/ddd/dddd, y/yy/yyyy/yyyy, h/hh/H/HH, m/mm, s/ss are supported.
<b>System.TeamProject</b>	The name of the team project to which this build belongs.
<b>Release.ReleaseId</b>	The ID of the release, which is unique across all releases in the project.
<b>Release.DefinitionName</b>	The name of the release definition to which the current release belongs.
<b>Build.BuildNumber</b>	The number of the build contained in the release. If a release has multiple builds, this is the number of the <a href="#">primary build</a> .

VARIABLE	DESCRIPTION
<b>Build.DefinitionName</b>	The definition name of the build contained in the release. If a release has multiple builds, this is the definition name of the <a href="#">primary build</a> .
<b>Artifact.ArtifactType</b>	The type of the artifact source linked with the release. For example, this can be <b>Team Build</b> or <b>Jenkins</b> .
<b>Build.SourceBranch</b>	The branch of the <a href="#">primary artifact source</a> . For Git, this is of the form <b>master</b> if the branch is <b>refs/heads/master</b> . For Team Foundation Version Control, this is of the form <b>branch</b> if the root server path for the workspace is <b>\$/teamproject/branch</b> . This variable is not set for Jenkins or other artifact sources.
<i>Custom variable</i>	The value of a global configuration property defined in the release definition.

For example, the release name format `Release $(Rev:rrr) for build $(Build.BuildNumber) $(Build.DefinitionName)` will create releases with names such as **Release 002 for build 20170213.2 MySampleAppBuild**.

## Release retention

You can customize how long releases of this definition must be retained. For more information, see [release retention](#).

## Release history

Every time you save a release definition, Release Management keeps a copy of the changes. This allows you to compare the changes at a later point, especially when you are debugging a deployment failure.

## Related topics

- [Artifacts](#)
- [Environments](#)
- [Triggers](#)
- [Variables](#)
- [Release retention](#)
- [Release security](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Environments in Release Management

2/26/2018 • 8 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

An **environment** is a *logical* and *independent* entity that represents where you want to deploy a release generated from a release definition. We'll examine these two characteristics in more detail to help you understand how to divide your release process into environments.

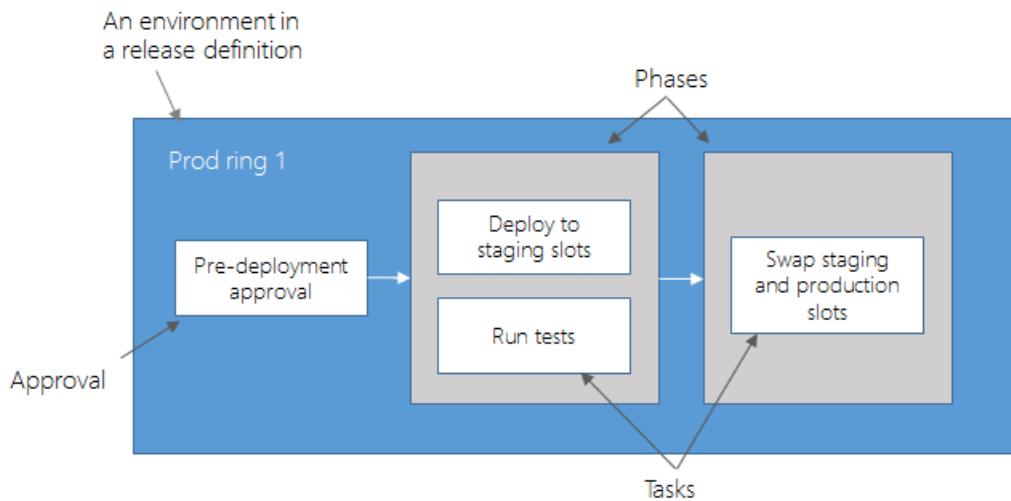
First, an environment in a release definition is a **logical** entity. It can represent any physical or real environment that you need. For example, the deployment in an environment may be to a collection of servers, a cloud, or multiple clouds. In fact, you can even use an environment to represent shipping the software to an app store, or the manufacturing process of a boxed product.

Second, you must be able to deploy to an environment **independently** of other environments in the definition. For example, your definition might consist of two environments A and B, and Release Management could deploy Release 2 to A and Release 1 to B. If you make any assumptions in B about the existence of a certain release in A, the two environments are not independent.

Here are some suggestions and examples for environments:

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

The deployment process of a release to an environment is defined in terms of [phases](#) and [tasks](#). The physical deployment of a release to an environment is controlled through [approvals and gates](#), [deployment conditions and triggers](#), and [queuing policies](#).



## Queuing policies

In some cases, you may be generating builds more quickly than they can be deployed. Alternatively, you may configure multiple [agents](#) and, for example, be creating releases from the same release definition for deployment of different artifacts. In such cases, it's useful to be able to control how multiple releases are queued into an environment. **Queuing policies** give you that control.

The screenshot shows the Azure DevOps Pipeline configuration for the "Fabrikam" project. The "Pipeline" tab is selected. On the left, there are sections for "Artifacts" and "Environments". The "Environments" section shows a "Dev" environment with "1 phase, 2 tasks". On the right, under "Pre-deployment conditions", there are sections for "Triggers", "Pre-deployment approvers", and "Deployment queue settings". The "Deployment queue settings" section is highlighted with a red box and contains options for "Number of parallel deployments" (set to "Specific" with value "5") and "Subsequent releases" (set to "Deploy all in sequence").

The options you can choose for a queuing policy are:

- **Maximum number of deployments that can proceed at one time:** Use this option if you dynamically provision new resources in your environment and it is physically capable of handling the deployment of multiple releases in parallel, but you want to limit the number of parallel deployments.

- If you specify a maximum number of deployments, two more options appear:
  - **Deploy all of them in order of request:** Use this option if you want to deploy all the releases sequentially into the same shared physical resources. By deploying them in turn, one after the other, you ensure that two deployment jobs do not target the same physical resources concurrently, even if there are multiple build and release agents available. You also ensure that pre-deployment approval requests for the environment are sent out in sequence.
  - **Deploy only the latest request and cancel the older ones:** Use this option if you are producing releases faster than builds, and you only want to deploy the latest build.

To understand how these options work, consider a scenario where releases **R1, R2, ..., R5** of a single release definition are created in quick succession due to new builds being produced rapidly. Assume that the first environment in this definition is named **QA** and has both pre-deployment and post-deployment approvers defined.

- If you do not specify a limit for the number of parallel deployments, all five approval requests will be sent out as soon as the releases are created. If the approvers grant approval for all of the releases, they will all be deployed to the **QA** environment in parallel. (if the **QA** environment did not have any pre-deployment approvers defined, all the five releases will automatically be deployed in parallel to this environment).
- If you specify a limit and **Deploy all of them in order of request**, and the limit has already been reached, the pre-deployment approval for release **R1** will be sent out first. After this approval is completed, the deployment of release **R1** to the **QA** environment begins. Next, a request for post-deployment approval is sent out for release **R1**. It is only after this post-deployment approval is completed that execution of release **R2** begins and its pre-deployment approval is sent out. The process continues like this for all of the releases in turn.
- If you specify a limit and **Deploy only the latest request and cancel the older ones**, and the limit has already been reached, releases **R2, R3, and R4** will be skipped, and the pre-deployment approval for **R5** in the **QA** environment will be sent out immediately after the post-deployment approval for release **R1** is completed.

## Environment general options

While the most important part of defining an environment is the automation tasks, you can also configure several properties and options for an environment in a release definition. You can:

- Edit the name of the environment here if required.
- Designate a single user or a single group to be the environment owner. Environment owners are notified whenever a deployment of a release is completed to that environment. Environment owners are not automatically assigned any addition permissions.
- Prevent the user who created a release or started the deployment from approving his or her own release. This is often useful to ensure compliance with corporate audit requirements.
- Force the identity of the user to be re-evaluated before the approval is processed and accepted.
- Delete the environment from the pipeline.
- Save a copy of the environment as a template.
- Manage the security settings for the environment.

The screenshot shows the Azure DevOps Pipeline interface. On the left, there's a sidebar with 'Environments' and a '+ Add' button. A specific environment named 'Dev' is selected and highlighted with a red box. On the right, the 'Environment' details are shown for 'Dev'. The 'Properties' section includes fields for 'Environment name' (set to 'Dev') and 'Environment owner' (set to 'Mateo Escobedo'). The 'Policies' section contains two checkboxes: 'User requesting a release or deployment should not approve the deployment to this environment' and 'Revalidate identity of approver before completing the approval.' A context menu is open over the environment card, showing options like 'Delete', 'Save as template', and 'Security'.

## Q&A

### Why do we call these *environments* instead of *stages*?

Primarily, to communicate the intent that they must be independent in a release definition. We felt that this term implies that you can have two different releases (sets of build artifacts) deployed to two environments within a definition at the same time. Release Management also allows you to deploy to multiple environments in parallel, and to directly deploy to any environment within the definition provided you have appropriate permission. It also helps you answer the question "Which release is currently deployed to a specific environment?".

"Stage" would be a more appropriate term if we implemented a model where a release, once created, must start in the first stage and complete all the stages before the next release can be created. Stages usually imply more dependencies. In other words, what you do in the second stage could be dependent on what you did in the first stage.

### Isn't environment more of a physical entity than a logical entity?

Unfortunately, the term environment is often also used to represent the real set of resources that are being deployed to, and this does cause some confusion with our choice of the term. We are exploring ways of clarifying this, or other alternatives to the term "environment".

### I need to deploy two Azure resource groups in order to deploy my application. Is that one environment or two?

An environment is a logical entity that represents an independent unit of deployment for your application, so you can deploy both the resource groups using a single environment. For more guidance on environments see the [introductory section](#) above.

### At the end of my pipeline, I update the binaries in an app store. I really do not have any environment in this case. How do I model this in a release definition?

An environment is a logical entity that can be used to perform any automation. It doesn't need to map to any physical resources. Therefore, you can add an environment to your release definition and add tasks to it to upload your binaries to the app store.

## Related topics

- [Environment triggers](#)
- [Tasks](#)
- [Environment security](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Artifacts in Release Management

2/26/2018 • 16 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

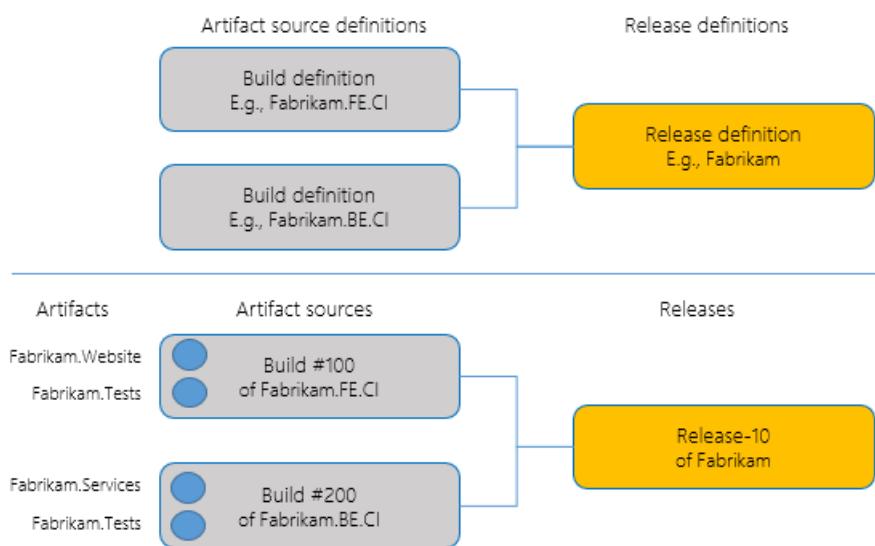
A release is a collection of artifacts. An **artifact** is a deployable component of your application. Release Management can deploy artifacts that are produced by a [wide range of artifact sources](#), and stored in different types of artifact repositories.

When **authoring a release definition**, you link the appropriate **artifact sources** to your release definition. For example, you might link a Team Build build definition or a Jenkins project to your release definition.

When **creating a release**, you specify the exact version of these artifact sources; for example, the number of a build coming from Team Build, 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 environments, you will be deploying and validating the same artifacts in all environments.

A single release definition can be linked to **multiple artifact sources**. 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 Release Management. Some of the features that depend on the linking of artifacts to a release definition are:

- **Auto-trigger releases.** You can configure new releases to be automatically created whenever a new version of an artifact is produced. For more details, see [Continuous deployment triggers](#). Note that the ability to automatically create releases is available for only some artifact sources.
- **Artifact variables.** Every artifact that is part of a release has metadata associated with it. 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 definition that produced the artifact (in the case of build artifacts), and more. This information is accessible in the deployment tasks. For more details, see [Artifact variables](#).
- **Work items and commits.** The work items or commits that are part of a release are computed from the versions of artifacts. For example, each build in Team Build 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

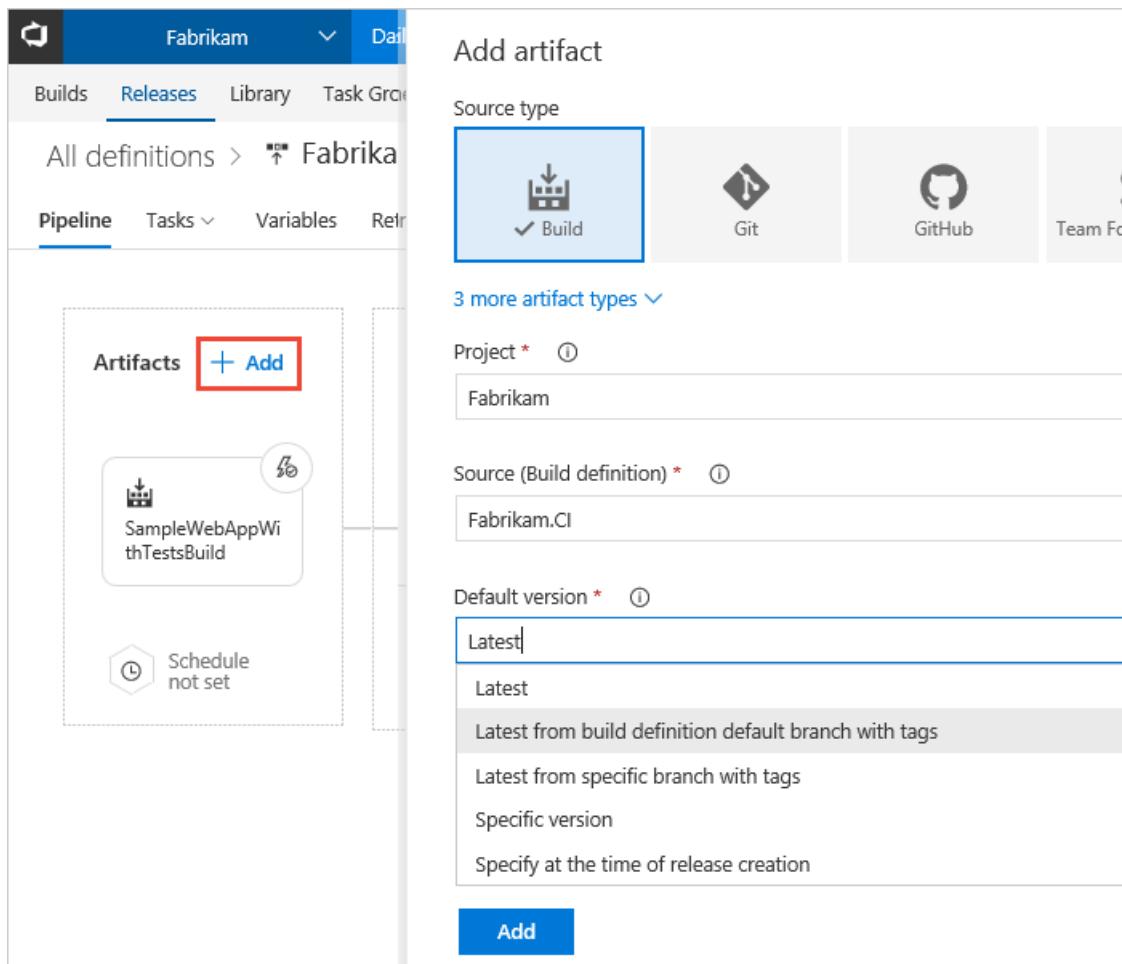
of all builds between the current release and the previous release. Note that Release Management is currently able to compute work items and commits for only certain artifact sources.

- **Artifact download.** Whenever a release is deployed to an environment, Release Management 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, Team Build artifacts are downloaded using an algorithm that downloads multiple files in parallel. Git artifacts are downloaded using Git library functionality. For more details, see [Artifact download](#).

## Artifact sources

There are several types of tools you might use in your application lifecycle process to produce or store artifacts. For example, you might use continuous integration systems such as Team Build, Jenkins, or TeamCity to produce artifacts. You might also use version control systems such as Git or TFVC to store your artifacts. You can configure Release Management to deploy artifacts from all these sources.

By default, a release created from the release definition will use the latest version of the artifacts. At the time of linking an artifact source to a release definition, 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 definition.

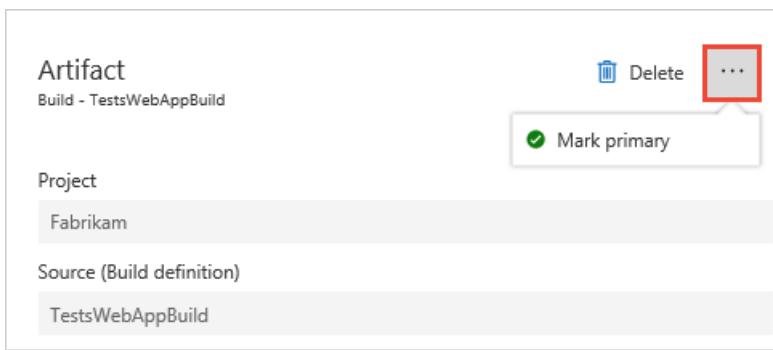


The screenshot shows the 'Add artifact' dialog in the Azure DevOps interface. On the left, the main pipeline view shows a single artifact named 'SampleWebAppWithTestsBuild'. On the right, the 'Add artifact' dialog is open, with the 'Build' source type selected. The dialog fields are as follows:

- Project:** Fabrikam
- Source (Build definition):** Fabrikam.CI
- Default version:** Latest

The 'Latest' option is highlighted in blue, indicating it is the selected default. Other options shown include 'Latest from build definition default branch with tags', 'Latest from specific branch with tags', 'Specific version', and 'Specify at the time of release creation'.

If you link more than one set of artifacts, you can specify which is the primary (default).



The following sections describe how to work with the different types of artifact sources.

## Team Build

You can link a release definition to any of the build definitions in your Visual Studio Team Services (VSTS) account or Team Foundation Server project collection.

### NOTE

you must include a **Publish Artifacts** task step in your build definition. For XAML build definitions, an artifact with the name **drop** is published implicitly.

Some of the differences in capabilities between different versions of TFS and VSTS are:

- **TFS 2015:** You can link build definitions only from the same team 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 definitions 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 VSTS:** You can link build definitions from any of the team projects in your collection or account. You can link multiple build definitions 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 Team Build sources:

FEATURE	BEHAVIOR WITH TEAM BUILD 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 definition. 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 Team Build.
Work items and commits	Team Build integrates with work items in TFS and VSTS. These work items are also shown in the details of releases. Team Build integrates with a number of version control systems such as TFVC and Git, GitHub, Subversion, and external Git repositories. Release Management 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 environment to <a href="#">skip the download</a> of artifacts.

FEATURE	BEHAVIOR WITH TEAM BUILD SOURCES
Deployment section in build	The build summary includes a <b>Deployment</b> section, which lists all the environments to which the build was deployed.

## Jenkins

To consume Jenkins artifacts, you must create a service endpoint with credentials to connect to your Jenkins server. For more details, see [service endpoints](#) and [Jenkins service endpoint](#). You can then link a Jenkins project to a release definition. 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 definition. 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	Release Management 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 environment 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 definition. Deployments download the artifacts automatically from Azure to the agents. In this configuration, connectivity between the agent and the Jenkins server is not required. Hosted agents can be used without exposing the server to internet.

### NOTE

Release Management in VSTS may not be able to contact your Jenkins server if, for example, it is within your enterprise network. In this case you can integrate Release Management 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 [VSTS Integration with Jenkins Jobs, Pipelines, and Artifacts](#).

## 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 process. For example:

- You are developing a PHP or a JavaScript application that does not require an explicit build process.
- You manage configurations for various environments in different version control repositories, and you want to consume these configuration files directly from version control as part of the deployment process.
- 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 definition, you can link both a build definition that produces the binaries of the application as well as a version control repository that stores the configuration files into the same definition, and use the two sets of artifacts together while deploying.

Release Management integrates with Team Foundation Version Control (TFVC) repositories, Git repositories, and GitHub repositories.

You can link a release definition to any of the Git or TFVC repositories in any of the team projects in your collection (you will need read access to these repositories). No additional setup is required when deploying version control artifacts within the same collection.

When you link a **Git** or **GitHub** repository and select a branch, you can edit the default properties of the artifact types after the artifact has been saved. This is particularly useful in scenarios where the branch for the stable version of the artifact changes, and continuous delivery releases should use this branch to obtain newer versions of the artifact. You can also specify details of the checkout, such as whether check out submodules and LFS-tracked files, and the shallow fetch depth.

When you link a **TFVC branch**, you can specify the changeset to be deployed when creating a release.

The following features are available when using TFVC, Git, and GitHub sources:

FEATURE	BEHAVIOR WITH TFVC, GIT, AND GITHUB SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for pushes into the repository in a release definition. 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	Release Management 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 environment to <a href="#">skip the download</a> of artifacts.

## Azure Container Registry, Docker, Kubernetes

When deploying containerized apps, the container image is first pushed to a container registry. After the push is complete, the container image can be deployed to the Web App for Containers service or a Docker/Kubernetes cluster. You must create a service endpoint with credentials to connect to your service to deploy images located there, or to your Azure account. For more details, see [service endpoints](#).

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	Release Management cannot show work items or commits.

FEATURE	BEHAVIOR WITH DOCKER SOURCES
Artifact download	By default, builds are downloaded to the agent. You can configure an option in the environment to <a href="#">skip the download</a> of artifacts.

## TeamCity

To integrate with TeamCity, you must first install the [TeamCity artifacts for Release Management](#) extension from Visual Studio Marketplace.

To consume TeamCity artifacts, start by creating a service endpoint with credentials to connect to your TeamCity server (see [service endpoints](#) for details).

You can then link a TeamCity build configuration to a release definition. 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 definition. To create a new release automatically when a build is complete, add a script to your TeamCity project that invokes the Release Management 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	Release Management cannot show work items or commits for TeamCity builds.
Artifact download	By default, TeamCity builds are downloaded to the agent. You can configure an option in the environment to <a href="#">skip the download</a> of artifacts.

### NOTE

Release Management in VSTS may not be able to contact your TeamCity server if, for example, it is within your enterprise network. In this case you can integrate Release Management 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.

## External TFS

You can use the Release Management service in VSTS 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 Management capabilities in VSTS.

Using this mechanism, you can also deploy artifacts published in one VSTS account from another VSTS account, 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 Release Management](#) extension from Visual Studio

Marketplace. Then create a service endpoint with credentials to connect to your TFS server (see [service endpoints](#) for details).

You can then link a TFS build definition to your release definition. Choose **External TFS Build** in the **Type** list.

The following features are available when using external TFS sources:

FEATURE	BEHAVIOR WITH EXTERNAL TFS SOURCES
Auto-trigger releases	You cannot configure a continuous deployment trigger for external TFS sources in a release definition. To automatically create a new release when a build is complete, you would need to add a script to your build definition in the external TFS server to invoke REST APIs of Release Management 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	Release Management 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 environment to <a href="#">skip the download</a> of artifacts.

#### NOTE

Release Management in VSTS may not be able to contact an on-premises TFS server if, for example, it is within your enterprise network. In this case you can integrate Release Management 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 definitions when linking to a build, but you can type these into the link dialog fields. In addition, when you create a release, VSTS may not be able to query the TFS server for the build numbers. Instead, type the **Build ID** (not the build number) of the desired build into the appropriate field, or select the **Latest** build.

## 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 Release Management, you can start using it without waiting for support for a specific source type. Simply skip the linking of artifact sources in a release definition, and add custom tasks to your environments that download the artifacts directly from your source.

## Artifact download

When you deploy a release to an environment, the versioned artifacts from each of the sources are, by default, downloaded to the automation agent so that tasks running within that environment 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 definition when you initiate a release, and the artifacts are downloaded into that folder. The `$(System.DefaultWorkingDirectory)` variable maps to this folder.

Note that, at present, Release Management 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, Release Management cannot perform incremental downloads to the agent.

You can, however, instruct Release Management to [skip the automatic download](#) of artifacts to the agent for a specific phase and environment of the deployment if you wish. Typically, you will do this when the tasks in that phase do not require any artifacts, or if you implement custom code in a task to download the artifacts you require.

## Artifact source alias

To ensure the uniqueness of every artifact download, each artifact source linked to a release definition 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 definition in Team Build 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; depending on the type of the artifact source this will be the name of the build definition, job, project, or repository. You can edit the source alias from the artifacts tab of a release definition; for example, when you change the name of the build definition and you want to use a source alias that reflects the name of the build definition.

The source alias can contain only alphanumeric characters and underscores, and must start with a letter or an underscore

## Primary source

When you link multiple artifact sources to a release definition, one of them is designated as the primary artifact source. The primary artifact source is used to set a number of pre-defined [variables](#). It can also be used in [naming releases](#).

## Artifact variables

Release Management 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 definition, you can access information about each of these. For a list of all pre-defined artifact variables, see [variables](#).

## Contributed links and additional information

- [Code repo sources in Release Management](#)
- [TeamCity extension for Release Management](#)
- [Jenkins artifacts in Release Management](#)
- [External TFS artifacts for Release Management](#)

## Related topics

- [Release definitions](#)
- [Environments](#)

## Help and support

- See our [troubleshooting](#) page.

- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Triggers in Release Management

2/28/2018 • 4 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

You can configure when releases should be created, and when those releases should be deployed to environments. The former is configured through [release triggers](#), and the latter through [environment triggers](#) - both in a release definition.

## Release (continuous deployment) triggers

If you specify [certain types](#) of artifacts in a release definition, you can enable continuous deployment. This instructs Release Management to create new releases automatically when it detects new artifacts are available. At present this option is available only for Team Foundation Build artifacts and Git-based sources such as Team Foundation Git, GitHub, and other Git repositories.

The screenshot shows the VSTS Pipeline interface for a release definition named "Fabrikam". On the left, there's a sidebar with "Artifacts" and a "+ Add" button. A red box highlights the "+ Add" button. In the main pane, under "Continuous deployment trigger", the "Build: SampleWebAppWithTestsBuild" is selected. The "Enabled" toggle switch is turned on. Below it, the text "Creates release every time a new build is available." is displayed. Under "Build branch filters", there are two entries: "master" with tag "web" and "devtest" with tag "local". A red box highlights the "Add" button next to the "devtest" entry. Below the filters, there are options for "Branch filter" and "Build Definition's default branch".

If you have linked multiple Team Foundation Build artifacts to a release definition, you can configure continuous deployment for each of them. In other words, you can choose to have a release created automatically when a new build of any of those artifacts is produced.

You add build branch filters if you want to create the release only when the build is produced by compiling code from certain branches (only applicable when the code is in a TFVC, Git, or GitHub repository) or when the build has certain tags. These can be both include and exclude filters. For example, use **features/\*** to include all builds under the **features** branch. You can also include [custom variables](#) in a filter value.

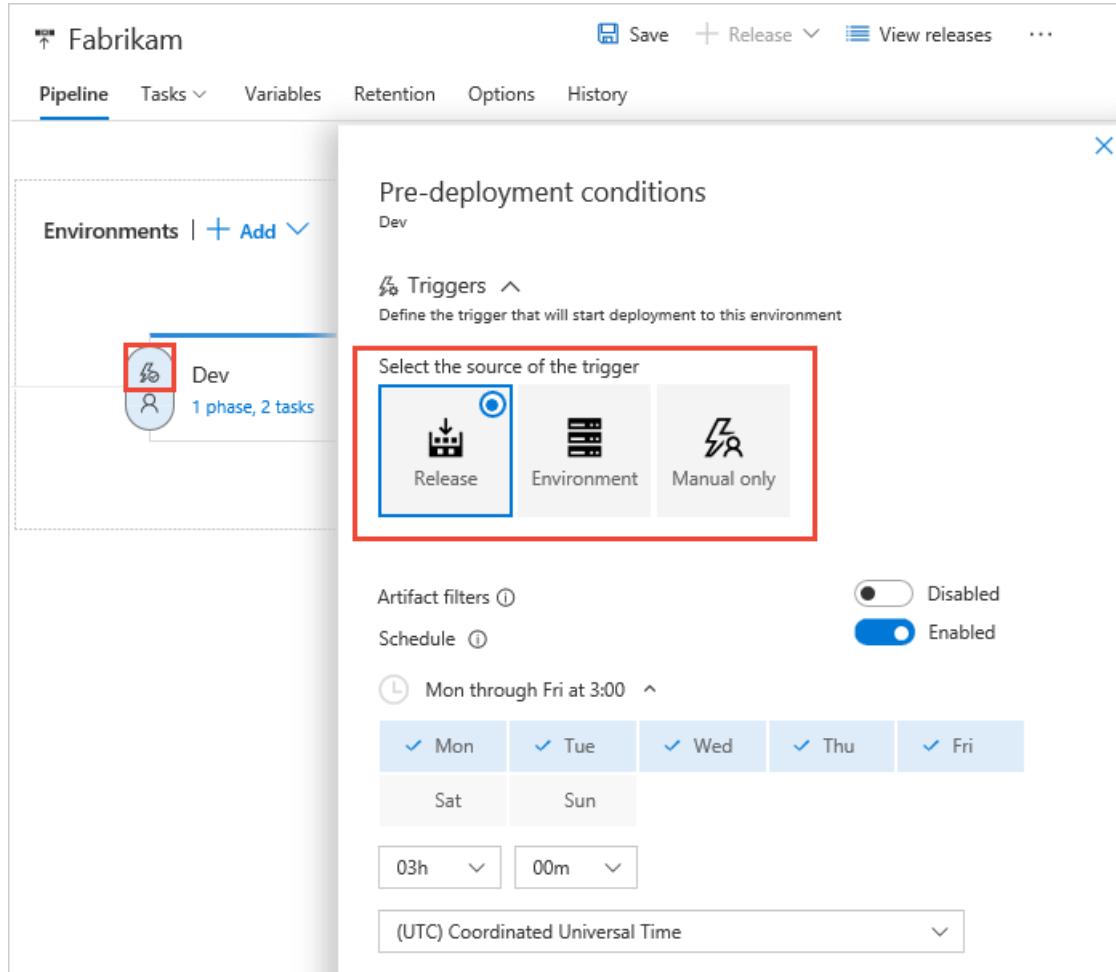
Alternatively, you can specify a filter to use the default branch specified in the build definition. This is useful when, for example, the default build branch changes in every development sprint. It means you don't need to update the trigger filter across all release definitions for every change - instead you just change the default branch in the build definition.

Note that, even though a release is automatically created, it might not be deployed automatically to any environments. The [environment triggers](#) govern when and if a release should be deployed to an environment.

## Environment triggers

You can choose to have the deployment to each environment triggered automatically when a release is created by a continuous deployment trigger, based on:

- **A predefined schedule.** When you select this option, you can select the days of the week and the time of day that Release Management will automatically create a new release. You can configure multiple schedules as required. Note that, with scheduled triggers, a new release is created even if a newer version of artifact is not available since the last release.



- **Filters based on the artifacts.** You can add one or more filters for each artifact linked to the release definition, and specify if you want to include or exclude particular branches of the code. Deployment will be triggered to this environment only if all the artifact conditions are successfully met.

Fabrikam

Pipeline Tasks Variables Retention Options History

Environments | + Add ▾

Dev

Triggers ▾ Define the trigger that will start deployment to this environment

Select the source of the trigger

Release Environment Manual only

Artifact filters ⓘ + Add ▾ Enabled

fabrikam-shell (1) fabrikam-core ^

Type Branch

Include live

+ Add

Pre-deployment conditions

- **The result of deploying to a previous environment in the pipeline.** Use this setting if you want the release to be first deployed and validated in another environment(s) before it is deployed to this environment. Triggers are configured for each environment, but the combination of these allows you to orchestrate the overall deployment - such as the sequence in which automated deployments occur across all the environments in a release definition. For example, you can set up a linear pipeline where a release is deployed first to the **Test** and **QA** environments. Then, if these two deployments succeed, it will be deployed to a **Staging** environment. In addition, you can configure the trigger to fire for partially succeeded (but not failed) deployments.

Fabrikam

Pipeline Tasks Variables Retention Options History

Staging

Triggers ▾ Define the trigger that will start deployment to this environment

Select the source of the trigger

Release Environment Manual only

Environment(s) that will trigger a deployment

QA, Test

Trigger when previous deployment partially succeeds

Artifact filters ⓘ Disabled

Schedule ⓘ Disabled

Pre-deployment conditions

- **Manually by a user.** Releases are not automatically deployed to the environment. To deploy a release to this environment, you must manually start a release and deployment from the release definition or from a build summary.

You can combine the automated settings to have releases created automatically either when a new build is available or according to a schedule.

**TFS 2015:** The following features are not available in TFS 2015 - continuous deployment triggers for multiple artifact sources, multiple scheduled triggers, combining scheduled and continuous deployment triggers in the same definition, continuous deployment based on the branch or tag of a build.

### Parallel forked and joined deployments

The **Triggering environment** list lets you select more than one environment. This allows you to configure parallel (*forked* and *joined*) deployment pipelines where the deployment to an environment occurs only when deployment to **all** the selected environments succeeds.

For example, the following schematic shows a pipeline where deployment occurs in parallel to the **QA** and **Pre-prod** environments after deployment to the **Dev** environment succeeds. However, deployment to the **Production** environment occurs only after successful deployment to both the **QA** and **Pre-prod** environments.



In combination with the ability to define [pre- and post-deployment approvals](#), this capability enables the configuration of complex and fully managed deployment pipelines to suit almost any release scenario.

Note that you can always deploy a release directly to any of the environments in your release definition by selecting the **Deploy** action when you create a new release. In this case, the environment triggers you configure, such as a trigger on successful deployment to another environment, do not apply. The deployment occurs irrespective of these settings. This gives you the ability to override the release process. Performing such direct deployments requires the **Manage deployments** permission, which should only be given to selected and approved users.

**TFS 2015:** Parallel fork and joined deployments are not available in TFS 2015

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Variables in Release Management

2/26/2018 • 11 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

As you compose the tasks for deploying your application into each environment, variables will help you to:

- Define a more generic deployment process once, and then customize it easily for each environment. 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 environment to another. These are **custom variables**.
- Use information about the context of the particular release, [environment](#), [artifacts](#), or [agent](#) in which the deployment process is being run. For example, your script may need access to the location of the build to download it, or to the working directory on the agent to create temporary files. These are **default variables**.

You can also use a default variable to [run a release in debug mode](#).

## Custom variables

Custom variables can be defined at various scopes.

- Share values across all of the definitions in a project by using [variable groups](#). Choose a variable group when you need to use the same values across all the definitions, environments, 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 environments by using **release definition variables**. Choose a release definition variable when you need to use the same value across all the environments and tasks in the release definition, 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 definition.
- Share values across all of the tasks within one specific environment by using **environment variables**. Use an environment-level variable for values that vary from environment to environment (and are the same for all the tasks in an environment). You define and manage these variables in the **Variables** tab of an environment in a release definition.

Using custom variables at project, release definition, and environment 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 definitions. Designate a configuration property to be a secure (secret) variable by selecting the  (padlock) icon next to the variable.

The values of hidden (secret) variables are stored securely on the server and cannot be viewed by users after they are saved. During a deployment, the Release Management service decrypts these values when referenced by the tasks and passes them to the agent over a secure HTTPS channel.

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)`.

You can use custom variables to prompt for values during the execution of a release. For more details, see

## Approvals.

### Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command.

#### 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 `secretSauce` variables

```
@echo ##vso[task.setvariable variable=sauce]crushed tomatoes  
@echo ##vso[task.setvariable variable=secretSauce;issecret=true]crushed tomatoes with garlic
```

 Read the variables

#### Arguments

```
"$(sauce)" "$(secretSauce)"
```

#### Script

```
@echo off  
set sauceArgument=%~1  
set secretSauceArgument=%~2  
@echo No problem reading %sauceArgument% or %SAUCE%  
@echo But I cannot read %SECRETSAUCE%  
@echo But I can read %secretSauceArgument% (but the log is redacted so I do not spoil  
the secret)
```

#### Console output from reading the variables:

```
No problem reading crushed tomatoes or crushed tomatoes  
But I cannot read  
But I can read ***** (but the log is redacted so I do not spoil the secret)
```

## Default variables

Information about the execution context is made available to running tasks through default variables. Your tasks and scripts can use these variables to find information about the system, release, environment, 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.

### System variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.TeamFoundationServiceUri	The URL of the Release Management service endpoint in the TFS or VSTS account. Use this from your scripts or tasks to call REST APIs on the Release Management service.	https://fabrikam.vssrm.visualstudio.com/	
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or VSTS account. Use this from your scripts or tasks to call REST APIs on other services such as Build and Version control.	https://fabrikam.visualstudio.com/	
System.CollectionId	The ID of the collection to which this build or release belongs.	6c6f3423-1c84-4625-995a-f7f143a1e43d	TFS 2015
System.TeamProject	The name of the team project to which this build or release belongs.	Fabrikam	
System.TeamProjectId	The ID of the team project to which this build or release belongs.	79f5c12e-3337-4151-be41-a268d2c73344	TFS 2015
System.ArtifactsDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
System.DefaultWorkingDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.ArtifactsDirectory.	C:\agent_work\r1\a	
System.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.	C:\agent_work	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.Debug	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.	true	

## Release variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.DefinitionName	The name of the release definition to which the current release belongs.	fabrikam-cd	
Release.DefinitionId	The ID of the release definition to which the current release belongs.	1	TFS 2015
Release.ReleaseName	The name of the current release.	Release-47	
Release.ReleaseId	The identifier of the current release record.	118	
Release.ReleaseUri	The URI of current release.	vstfs:///ReleaseManagement/Release/118	
Release.ReleaseDescription	The text description provided at the time of the release.	Critical security patch	
Release.RequestedFor	The display name of identity that triggered the release.	Mateo Escobedo	
Release.RequestedForId	The ID of identity that triggered the release.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	
Release.EnvironmentName	The name of environment to which deployment is currently in progress.	Dev	
Release.EnvironmentId	The ID of the environment instance in a release to which the deployment is currently in progress.	276	
Release.EnvironmentUri	The URI of environment instance in a release to which deployment is currently in progress.	vstfs:///ReleaseManagement/Environment/276	
Release.DefinitionEnvironmentId	The ID of the environment in the corresponding release definition.	1	TFS 2015

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.AttemptNumber	The number of times this release is deployed in this environment.	1	TFS 2015
Release.Deployment.RequestedFor	The display name of the identity that triggered (started) the deployment currently in progress.	Mateo Escobedo	TFS 2015
Release.Deployment.RequestedForId	The ID of the identity that triggered (started) the deployment currently in progress.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	TFS 2015

## Release environment variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.Environments.{Environment name}.Status	The status of deployment of this release within a specified environment.	NotStarted	TFS 2015

## Agent variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
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.	fabrikam-agent	
Agent.MachineName	The name of the computer on which the agent is configured.	fabrikam-agent	
Agent.Version	The version of the agent software.	2.109.1	
Agent.JobName	The name of the job that is running, such as Release or Build.	Release	
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.	C:\agent	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.ReleaseDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as System.ArtifactsDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
Agent.RootDirectory	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.WorkFolder and System.WorkFolder.	C:\agent_work	
Agent.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and System.WorkFolder.	C:\agent_work	
Agent.DeploymentGroupId	The ID of the deployment group the agent is registered with. This is available only in deployment group phases.	1	TFS 2018 U1

## Artifact variables

For each artifact that is referenced in a release, you can use the following artifact variables. Not all variables are meaningful for each artifact type. The table below lists the default artifact variables and provides examples of the values that they have depending on the artifact type. If an example is empty, it implies that the variable is not populated for that artifact type.

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{Artifact alias}.DefinitionId	The identifier of the build definition or repository.	1			fabrikam/asp
Release.Artifacts.{Artifact alias}.DefinitionName	The name of the build definition or repository.	fabrikam-ci		TFVC: \$/fabrikam, Git: fabrikam	fabrikam/asp (master)
Release.Artifacts.{Artifact alias}.BuildNumber	The build number or the commit identifier.	20170112.1	20170112.1	TFVC: Changeset 3, Git: 38629c964	38629c964

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/ TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{Artifact alias}.BuildId	The build identifier.	130	130		38629c964d21fe 405ef830b7d022 0966b82c9e11
Release.Artifacts.{Artifact alias}.BuildURI	The URL for the build.	vstfs:///build-release /Build/130			
Release.Artifacts.{Artifact alias}.SourceBranch	The path of the branch from which the source was built.	refs/heads/master			
Release.Artifacts.{Artifact alias}.SourceBranchName	The name of the branch from which the source was built.	master			
Release.Artifacts.{Artifact alias}.SourceVersion	The commit that was built.	bc0044458ba1d9 298 cdc649cb5dcf013 180706f7			
Release.Artifacts.{Artifact alias}.Repository.Provider	The type of repository from which the source was built	Git			
Release.Artifacts.{Artifact alias}.RequestedForID	The identifier of the account that triggered the build.	2f435d07-769f-4e46-849d-10d1ab9ba6ab			
Release.Artifacts.{Artifact alias}.RequestedFor	The name of the account that requested the build.	Mateo Escobedo			
Release.Artifacts.{Artifact alias}.Type	The type of artifact source, such as Build.	Build	Jenkins: Jenkins, TeamCity: TeamCity	TFVC: TFVC, Git: Git	GitHub

See also [Artifact source alias](#)

## Primary artifact variables

You designate one of the artifacts as a primary artifact in a release definition. For the designated primary artifact, Release Management populates the following variables.

VARIABLE NAME	SAME AS
Build.DefinitionId	Release.Artifacts.{Primary artifact alias}.DefinitionId
Build.DefinitionName	Release.Artifacts.{Primary artifact alias}.DefinitionName

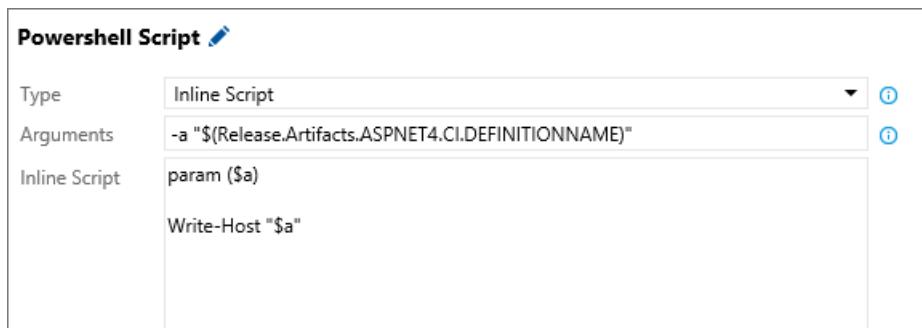
VARIABLE NAME	SAME AS
Build.BuildNumber	Release.Artifacts.{Primary artifact alias}.BuildNumber
Build.BuildID	Release.Artifacts.{Primary artifact alias}.BuildId
Build.BuildURI	Release.Artifacts.{Primary artifact alias}.BuildURI
Build.SourceBranch	Release.Artifacts.{Primary artifact alias}.SourceBranch
Build.SourceBranchName	Release.Artifacts.{Primary artifact alias}.SourceBranchName
Build.SourceVersion	Release.Artifacts.{Primary artifact alias}.SourceVersion
Build.Repository.Provider	Release.Artifacts.{Primary artifact alias}.Repository.Provider
Build.RequestedForID	Release.Artifacts.{Primary artifact alias}.RequestedForID
Build.RequestedFor	Release.Artifacts.{Primary artifact alias}.RequestedFor
Build.Type	Release.Artifacts.{Primary artifact alias}.Type

## Using default variables

You can use the default variables in two ways - as parameters to tasks in a release definition or in your scripts.

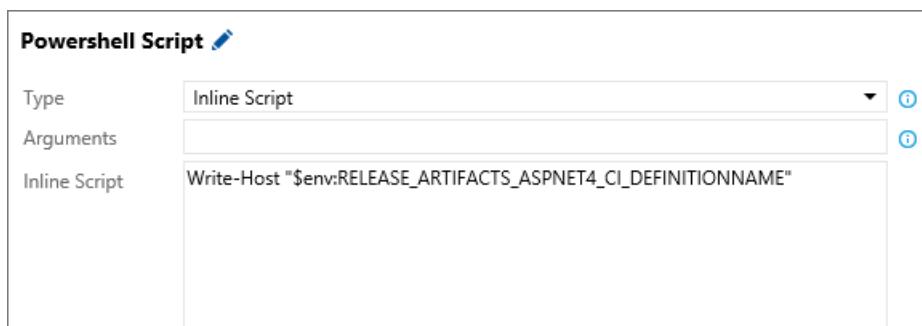
You can directly use a default variable as an input to a task. For example, to pass

`Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** to a task, you would use `$(Release.Artifacts.ASPNET4.CI.DefinitionName)`.



To use a default variable in your script, you must first replace the `.` in the default variable names with `_`. For example, to print the value of artifact variable `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** in a Powershell script, you would use

`$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME`.



Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

## 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 environment, 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 definition.
- To initiate debug mode for a single environment, open the **Configure environment** dialog from the shortcut menu of the environment 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 definition.

If you get an error related to an Azure RM service endpoint, see [How to: Troubleshoot Azure Resource Manager service endpoints](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Approvals and gates overview

2/28/2018 • 2 min to read • [Edit Online](#)

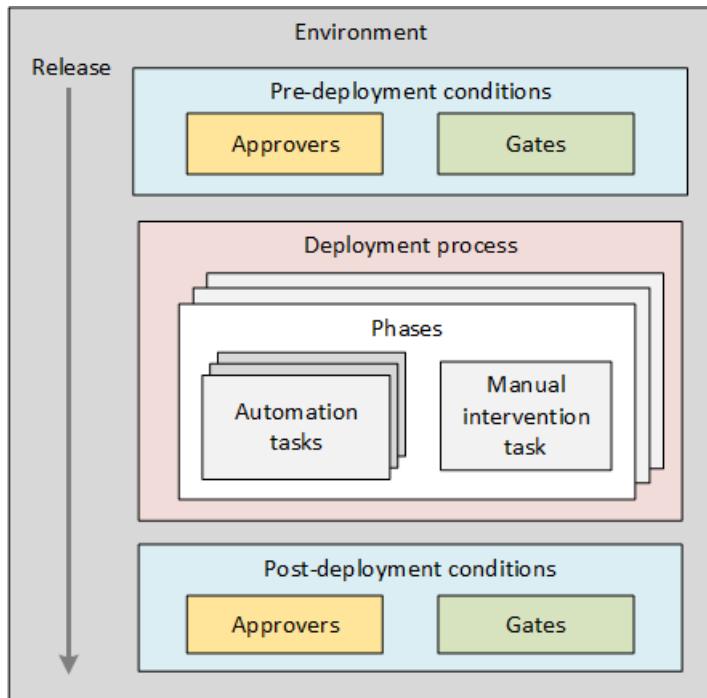
## VSTS | TFS 2018 | TFS 2017 | TFS 2015

A release definition specifies the end-to-end release process for an app to be deployed across a range of environments. Deployments to each environment are fully automated by using [phases](#) and [tasks](#).

**Approvals** and **gates** give you additional control over the start and completion of the deployment process. Each environment in a release definition can be configured with pre-deployment and post-deployment conditions that can include waiting for users to manually approve or reject deployments, and checking with other automated systems until specific conditions are verified. In addition, you can configure a manual intervention to pause the deployment process and prompt users to carry out manual tasks, then resume or reject the deployment.

At present, gates are available only in Visual Studio Team Services.

The following diagram shows how these features are combined in an environment of a release definition.



By using approvals, gates, and manual intervention you can take full control of your releases to meet a wide range of deployment requirements. Typical scenarios where approvals, gates, and manual intervention are useful include the following.

SCENARIO	FEATURE(S) TO USE
Some users must manually validate the change request and approve the deployment to an environment.	<a href="#">Pre-deployment approvals</a>
Some users must manually sign off the app after deployment before the release is promoted to other environments.	<a href="#">Post-deployment approvals</a>

SCENARIO	FEATURE(S) TO USE
You want to ensure there are no active issues in the work item or problem management system before deploying a build to an environment.	<a href="#">Pre-deployment gates</a>
You want to ensure there are no incidents from the monitoring or incident management system for the app after it's been deployed, before promoting the release.	<a href="#">Post-deployment gates</a>
After deployment you want to wait for a specified time before prompting some users for a manual sign-off.	<a href="#">Post-deployment gates</a> and <a href="#">post-deployment approvals</a>
During the deployment process a user must manually follow specific instructions and then resume the deployment.	<a href="#">Manual Intervention</a>
During the deployment process you want to prompt the user to enter a value for a parameter used by the deployment tasks, or allow the user to edit the details of this release.	<a href="#">Manual Intervention</a>
During the deployment process you want to wait for monitoring or information portals to detect any active incidents, before continuing with other deployment phases.	Planned

You can, of course, combine all three techniques within a release definition to fully achieve your own process and deployment requirements.

## Related topics

- [Approvals](#)
- [Gates](#)
- [Manual intervention](#)
- [Environments](#)
- [Triggers](#)
- [Release definitions and releases](#)

## See also

- [Video: Deploy quicker and safer with gates in VSTS](#)
- [Configure your release pipelines for safe deployments](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Approvals

2/28/2018 • 4 min to read • [Edit Online](#)

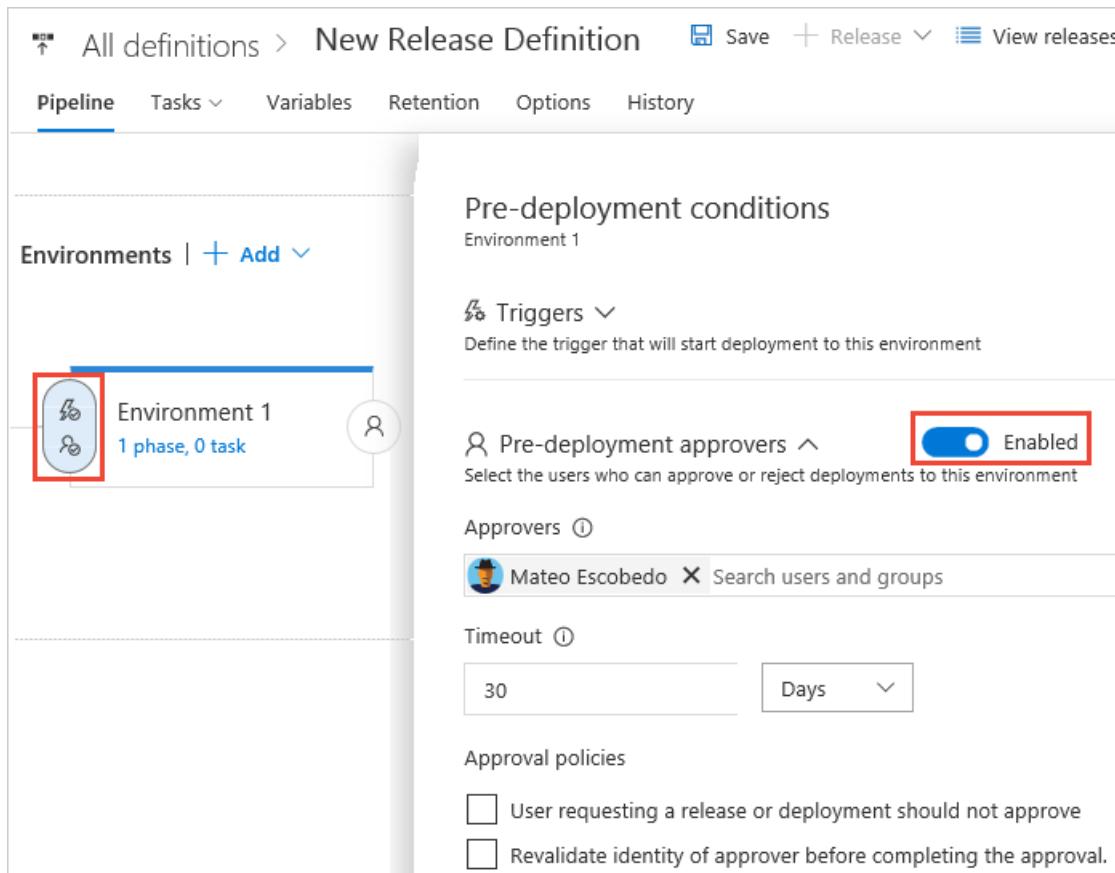
**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

When a release is created from a release definition 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 environment in a release definition.

## Define a deployment approval

1. Decide if you need pre-deployment approvers, post-deployment approvers, or both for an environment. Then open the appropriate conditions panel(s).

For a **pre-deployment** approval, choose the icon at the entry point of the environment and enable pre-deployment approvers.



The screenshot shows the 'New Release Definition' interface. On the left, there's a sidebar with 'Environments' and a '+ Add' button. The main area is titled 'Pre-deployment conditions' for 'Environment 1'. It includes sections for 'Triggers', 'Pre-deployment approvers' (which is enabled), 'Approvers' (listing 'Mateo Escobedo'), 'Timeout' (set to 30 days), and 'Approval policies' (with two unchecked checkboxes: 'User requesting a release or deployment should not approve' and 'Revalidate identity of approver before completing the approval').

For a **post-deployment** approval, choose the icon at the exit point of the environment and enable post-deployment approvers.

The screenshot shows the 'New Release Definition' interface in VSTS. On the left, there's a sidebar with 'Environments' and a button to 'Add'. In the main area, 'Environment 1' is selected, showing '1 phase, 0 tasks'. To the right, the 'Post-deployment conditions' section is open. It includes a search bar for approvers, a toggle switch labeled 'Enabled' (which is highlighted with a red box), a list of approvers (Mateo Escobedo, also highlighted with a red box), a 'Timeout' field set to 30 days, and an unchecked checkbox for approval policies.

2. Select one or more **Approvers** for the approval step. You can add multiple approvers for both pre-deployment and post-deployment settings. These approvers can be individual users or groups of users. When a group is specified as an approver, only one of the users in that group needs to approve for the deployment to occur or the release to move forward.

- If you are using **Visual Studio Team Services** (VSTS), you can use local groups managed in VSTS or Azure Active Directory (AAD) groups if they have been added into VSTS.
- 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.

3. Specify the **Timeout** for the approval. If no approval is granted within the timeout period you specify, the deployment is rejected.

4. Specify the **Approval policies** you require:

- You can 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.
- You can force a revalidation of the user identity to take into account recently changed permissions.
- You can reduce user workload by automatically approving subsequent prompts if the specified user has already approved the deployment to a previous environment 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 environment in the same release pipeline.

User requesting a release or deployment should not approve

Revalidate identity of approver before completing the approval.

Skip approval if the same approver approved the previous environment ①

## Approval notifications

Release Management can send notifications such as an email message to the approver(s) defined for each approval step.

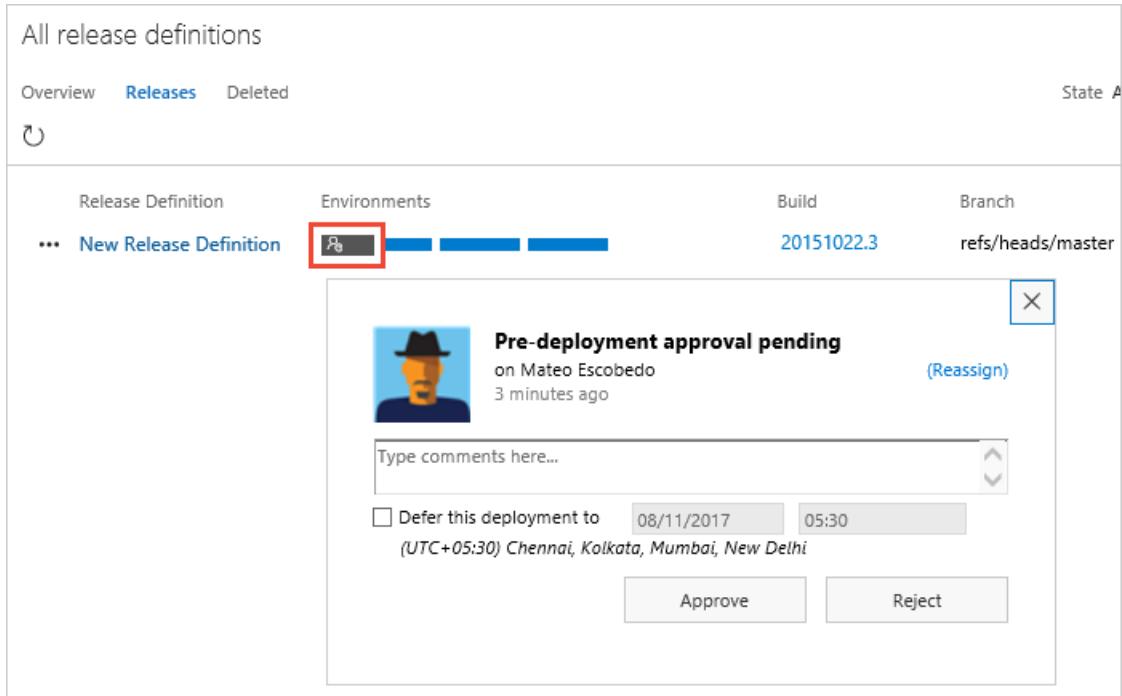
Category	Notification Type	Description
Build	Build completes	Build completes
	Build completed (any project)	(any project)
	Pull request reviewers added or removed	Notify the team when it is added or removed as a reviewer for a pull request
	Pull request changes	Notify the team when changes are made to a pull request the team is a reviewer for
	Manual intervention pending	Notify the team when a manual intervention is pending on the team
	Deployment to an owned environment failed	Notify the team when a deployment to an environment team owns fails to complete
Code (Git)	Deployment to an approved environment failed	Notify the team when a deployment team approved fails to complete successfully
	Deployment completion failures	Notify the team when a deployment team requested fails to complete successfully
	Deployment approval pending	Notify the team when an approval for a deployment is pending on the team
	Deployment pending (any project)	(any project)
	Deployment completed (any project)	(any project)
Release	Deployment completed (any project)	(any project)
	Deployment approval pending (any project)	(any project)
	Release approval pending (any project)	(any project)
	Release completed (any project)	(any project)

The link in the email message opens the **Summary** page for the release where the user can approve or reject the release.

## Approve or reject a deployment

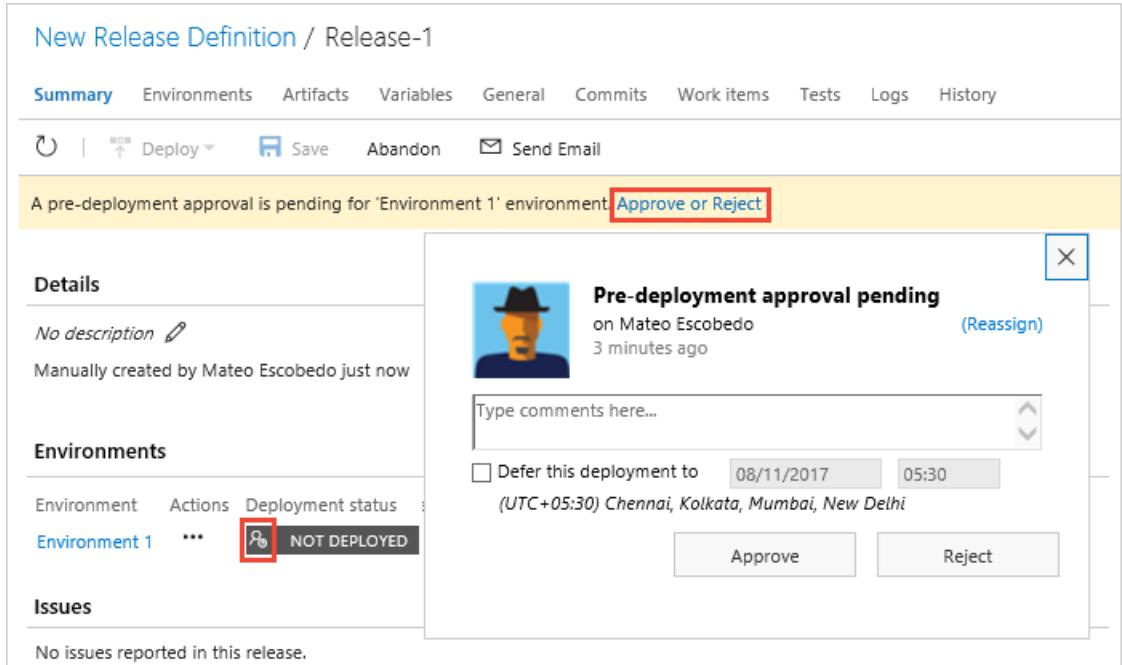
During a release, the deployment pipeline will stop at any stage that requires approval, and will display an alert or

indicator to the user. In the **Releases**, **Summary**, and **Logs** pages and lists, it displays the  icon when the deployment is waiting for approval by the current user, or the  icon when the deployment is waiting for approval by a different user. This link or icon displays the approval dialog.



The screenshot shows the 'All release definitions' page. At the top, there are tabs for 'Overview', 'Releases' (which is selected), and 'Deleted'. On the right, it says 'State A'. Below the tabs, there's a button labeled 'New Release Definition' and a 'Refresh' icon. The main area has sections for 'Release Definition', 'Environments', 'Build', and 'Branch'. The 'Environments' section shows a status bar with four colored segments (blue, red, green, blue) and a 'Pre-deployment approval pending' icon. The 'Build' section shows '20151022.3' and the 'Branch' section shows 'refs/heads/master'. A pop-up dialog box is overlaid on the page, titled 'Pre-deployment approval pending' for 'Mateo Escobedo' (3 minutes ago). It includes a comment input field, a date/time selector for deferring the deployment, and 'Approve' and 'Reject' buttons.

A notification bar is also shown in the release details page, with a link that displays the approval dialog.

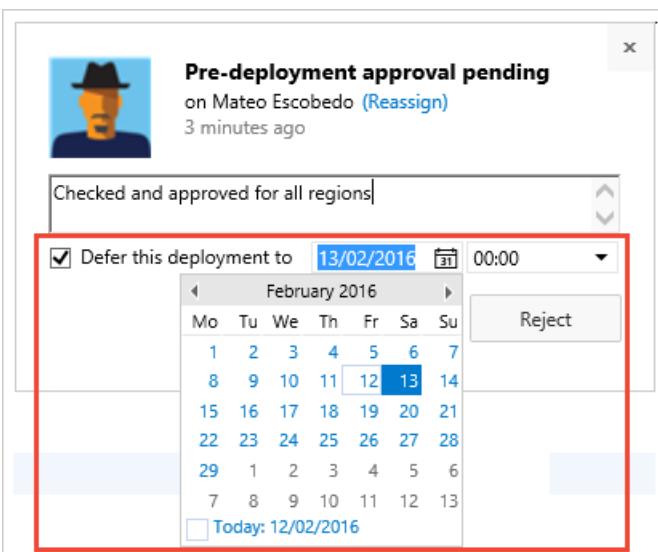


The screenshot shows the 'New Release Definition / Release-1' details page. At the top, there are tabs for 'Summary' (selected), 'Environments', 'Artifacts', 'Variables', 'General', 'Commits', 'Work items', 'Tests', 'Logs', and 'History'. Below the tabs are buttons for 'Deploy', 'Save', 'Abandon', and 'Send Email'. A yellow notification bar at the top states 'A pre-deployment approval is pending for 'Environment 1' environment' with a link 'Approve or Reject' (which is highlighted with a red box). The main content area has sections for 'Details' (with a 'No description' note and 'Manually created by Mateo Escobedo just now'), 'Environments' (listing 'Environment 1' with a status 'NOT DEPLOYED' and a 'Pre-deployment approval pending' icon), and 'Issues' (stating 'No issues reported in this release'). A pop-up dialog box is overlaid on the page, identical to the one in the previous screenshot, showing the approval details for 'Environment 1'.

If there is more than one approval waiting for the same user, the reminder will contain two links, **All** and **Selected environments**. This saves the user from needing to interact with each approval request individually.

Use the approvers pop-up dialog to:

- Enter a comment
- Approve or reject the deployment
- Reassign the approval to somebody else
- Defer the deployment to a specified date and time



An administrator can approve a deployment step even if the approval was originally defined for a different user. In this case, the approvers pop-up dialog contains an **Override** link instead of a **Reassign** link. This enables the administrator to enter comments, approve, reject, reassign, or defer the deployment.

You can also approve and reject pending deployments by accessing the [Release Management REST API](#).

## View and monitor manual approvals

During and after a deployment, the **Logs** page shows comprehensive information about the progress and status of the release. Pre-deployment and post-deployment items are included; choose the **Action** icon next to them to see the approval action log entry.

For example, in this screenshot pre-deployment approval was granted, and the **Action** link displays the name of the approver and the message entered by that user when approving (or rejecting) the deployment.

Notice that this release is now waiting for post-deployment approval. Choosing the **Action** item for this will display the approval dialog where the deployment can be approved or rejected.

## Related topics

- [Approvals and gates overview](#)
- [Manual intervention](#)

- [Environments](#)
- [Triggers](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Gates

2/28/2018 • 3 min to read • [Edit Online](#)

## VSTS

Gates allow you to query a range of external services, and wait for a positive input from all of them before continuing with a deployment to an environment. When a release is created from a definition that contains gates, the deployment stops until the health signals from all the configured services are successful.

You may need to enable Gates in your profile [Preview features](#) list.

## Define a gate for an environment

1. Enable gates in the **Pre-deployment conditions** or **Post-deployment conditions** panel for an environment.

The screenshot shows the 'New Release Definition' pipeline settings in VSTS. On the left, there's a sidebar with 'Environments' and a list containing 'Environment 1' (1 phase, 0 tasks). The main area is titled 'Pre-deployment conditions'. It includes sections for 'Triggers', 'Pre-deployment approvers' (disabled), and 'Gates'. The 'Gates' section is highlighted with a red box around its toggle switch, which is set to 'Enabled'. Below it, there's a 'Delay before evaluation' field set to 15 minutes. Other sections include 'Deployment gates' and 'Deployment queue settings'.

2. Specify the **Delay before evaluation** for all the gates you intend to use. This is a time delay at the beginning of the initial gate evaluation process that allows the gates to initialize, stabilize, and begin providing accurate results for the current deployment. See [Gate evaluation flows](#).

Gates ^

Enabled

Define gates to evaluate before the deployment. [Learn more](#)

Delay before evaluation \* ⓘ

15 Minutes

Deployment gates ⓘ + Add

As an 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 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 environment, and the least time it takes for incidents to be logged after the deployment.

3. Choose **+ Add**, and select the type of release gate you require.

Gates ^

Enabled

Define gates to evaluate before the deployment. [Learn more](#)

Delay before evaluation ⓘ

15 Minutes

Deployment gates ⓘ + Add

Deployment queue settings ⓘ

Define behavior when multiple releases are triggered.

Azure Monitor  
Observe the configured Azure monitor rules for active alerts.

Invoke Azure Function  
Invoke Azure Function as a part of your process.

Invoke REST API  
Invoke REST API as a part of your process.

Query Work Items  
Executes a work item query and checks for the number of items returned.

At present the available gates include:

- **Azure function**: Trigger execution of an Azure function and ensure a successful completion. For more details, see [Azure function task](#).
- **Azure monitor**: 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](#).
- **Work item query**: Ensure the number of matching work items returned from a query is within a threshold. For more details, see [Work item query task](#).

Also see the tutorial [Use approvals and gates to control your deployment](#) and the blog post [Twitter sentiment as a release gate](#), which includes an example of a gate that uses an Azure function. A [library with examples](#) is available to help you create your own custom gate tasks.

4. Select and enter the required gate arguments, depending on the type of gate you chose.

Deployment gates ⓘ

**Add**

**Query Work Items** Enabled ⚡

Query Work Items (Preview) ⓘ

Display name \*

Query Work Items

Query \* ⓘ

Select query

Maximum Threshold \* ⓘ

0

Advanced ^

Minimum Threshold \* ⓘ

0

5. Set the options that apply to all the gates you added:

- **Timeout.** 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.
- **Sampling interval.** The time interval between each evaluation of all the gates. At each sampling interval, new requests are sent concurrently to each gate for fresh results. The sampling interval must be greater than the longest typical response time of any configured gate to allow time for all responses to be received.
- **Execution order.** Select the required order of execution for gates and approvals if you have configured both. For pre-deployment conditions, the default is to prompt for manual (user) approvals first, then evaluate gates afterwards. This saves the system from evaluating the gate functions if the release is rejected by the user. For post-deployment conditions, the default is to evaluate gates and prompt for manual approvals only when all gates are successful. This ensures the approvers have all the information required for a sign-off.

Options for all gates ^

Timeout \* ⓘ

1 Days

Sampling interval \* ⓘ

15 Minutes

Execution order of approvals and gates ⓘ

Manual approvals before gates

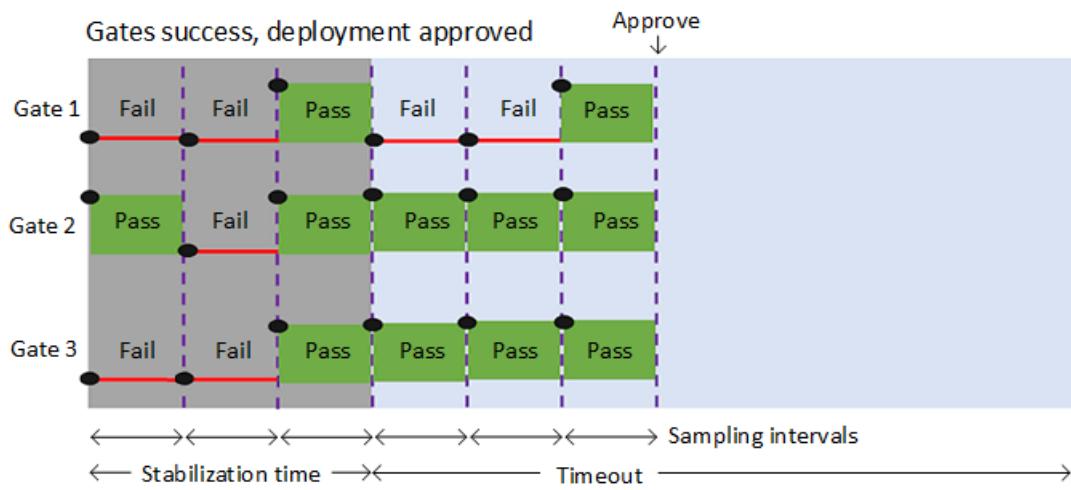
Manual approvals after all gates succeed

Manual approvals after gates - always

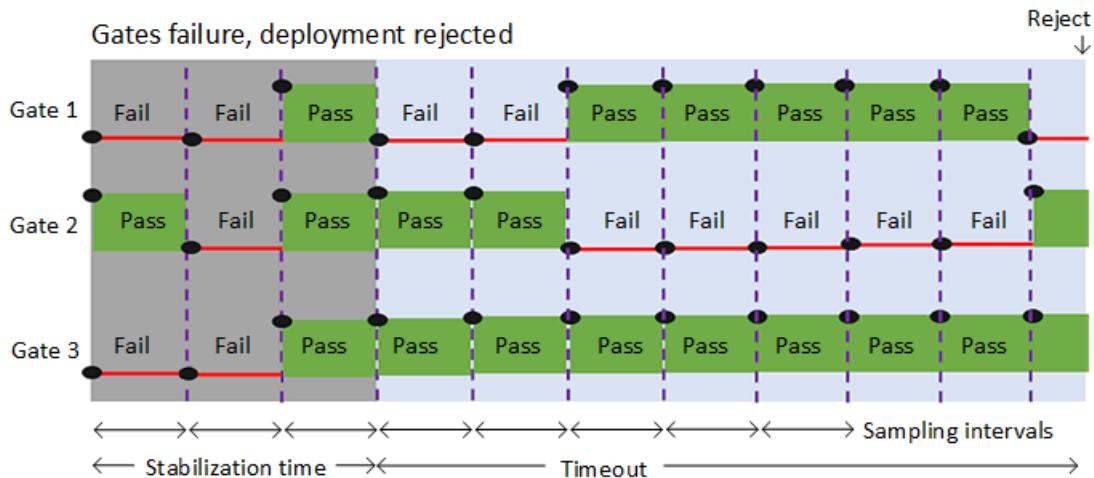
Watch [this video on Channel 9](#) to see gates in action.

### 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.



## View and monitor gate results

1. Open the **Summary** page for your release. As the release executes, the pop-up message when you choose the **?** icon for an environment indicates the current status of your deployment to each environment.

**New Release Definition / Release-2**

<b>Summary</b>	Environments	Artifacts	Variables	General	Commits	Work items	Tests	Logs	History
<span style="border: 1px solid #ccc; padding: 2px;">Deploy</span>   <span style="border: 1px solid #ccc; padding: 2px;">Save</span> <span style="border: 1px solid #ccc; padding: 2px;">Abandon</span> <span style="border: 1px solid #ccc; padding: 2px;">Send Email</span>									

**Details**

No description (edit)

Manually created by Mateo Escobedo just now

**Work items**

No associated work items found.

**Environments**

Enviro...	Actions	Deployment status	Triggered	Compl...	Tests
Enviro...	***	NOT DEPLOYED	(?)	just now	No tests

**Tags**

Add...

**Issues**

Stabilizing gates. View logs for more details.

No issues reported in this release.

2. Open the **Logs** page for your release. During and after a deployment, it shows comprehensive information about the evaluation of all the gates you configured for the release.

New Release Definition / Release-2

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

Deploy Save Abandon Download all logs as zip Send Email

Step	Action
Environment 1	...
Pre-deployment approval	
Pre-deployment gates	
Query Work Items	

**Pre-deployment gates summary**

Status	STABILIZING GATES ⓘ
Last sampling time	8 minutes ago
Next sampling time	07/11/2017 22:18
Stabilization start time	07/11/2017 22:03
Stabilization end time	07/11/2017 22:18

**Recent gate sampling ⓘ**

Query Work Items
<input checked="" type="checkbox"/>

Showing latest 1 of 1 samples. Download all logs to get results for all

## Related topics

- [Approvals and gates overview](#)
- [Manual intervention](#)
- [Use approvals and gates to control your deployment](#)
- [Environments](#)
- [Triggers](#)

## See also

- [Video: Deploy quicker and safer with gates in VSTS](#)
- [Configure your release pipelines for safe deployments](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Environment templates in Release Management

2/26/2018 • 2 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

When you start a new release definition, or when you add an environment to an existing release definition, you can choose from a list of templates for each environment. These templates pre-populate the environment with the appropriate tasks and settings, which can considerably reduce the time and effort required to create a release definition.

A set of pre-defined environment templates are available in VSTS and in each version of TFS. You can use these templates when you create a new release definition or add a new environment to a definition. You can also create your own custom environment templates from an environment you have populated and configured.

Templates do not have any additional security capability. There is no way to restrict the use of a template to specific users. All templates, pre-defined and custom, are available for use by all users who have permission to create release definitions.

When an environment is created from a template, the tasks in the template are copied over to the environment. Any further updates to the template have no impact on existing environments. If you want a way to easily insert a number of environments into release definitions (perhaps to keep the definitions consistent) and to enable these environments to all be updated in one operation, use [task groups](#) instead of environment templates.

## Q&A

### **Can I export templates or share them with other accounts or projects?**

Custom templates that you create are scoped to the team project that you created them in. Templates cannot be exported or shared with another team project, collection, server, or account. You can, however, export a release definition and import it into another project, collection, server, or account. Then you can re-create the template for use in that location.

### **Can I publish or consume new templates through extensions in VS Marketplace?**

Yes. See [Adding release management environment templates to your VSS extension](#) for more details.

### **How do I delete a custom environment template?**

You can delete an existing custom template from the list of templates that is displayed when you add a new environment to our definition.

## Select a Template

Or start with an [Empty process](#)

applications to IIS website.

### IIS IIS Website and SQL Database offline upgrade

Deployment Group: Upgrade ASP.Net, ASP.Net core based websites. Upgrade SQL database using SQL scripts executed when web application is offline.

### IIS IIS Website and SQL Database online upgrade

Deployment Group: Upgrade ASP.Net, ASP.Net core based websites. Upgrade SQL database using SQL scripts executed when web application is online.

### Custom



DevTemplate

Custom Dev environment template

Apply



## How do I update a custom environment template?

To update an environment template, delete the existing template in a release definition and then save the environment as a template with the same name.

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Releases in Release Management

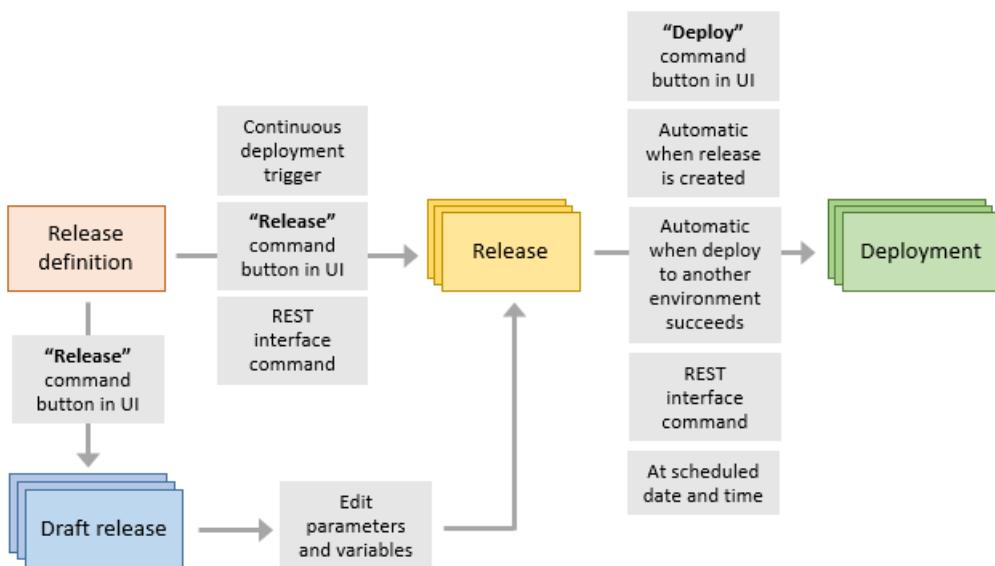
2/28/2018 • 2 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

A **release** is the package or container that holds a versioned set of artifacts specified in a [release definition](#). It includes a snapshot of all the information required to carry out all the tasks and actions in the release definition, such as the [environments](#), the task steps for each one, the values of task parameters and variables, and the release policies such as triggers, approvers, and release queuing options. There can be multiple releases from one released definition, and information about each one is stored and displayed in Release Management for the specified [retention period](#).

A **deployment** is the action of running the [tasks](#) for one environment, which results in the application [artifacts](#) being deployed, tests being run, and whatever other actions are specified for that environment. Initiating a release starts each deployment based on the settings and policies defined in the original release definition. There can be multiple deployments of each release even for one environment. When a deployment of a release fails for an environment, you can redeploy the same release to that environment.

The following schematic shows the relationship between release definitions, releases, and deployments.



Releases (and, in some cases, draft releases) can be created from a release definition in several ways:

- By a [continuous deployment trigger](#) that creates a release when a new version of the source build artifacts is available.
- By using the **Release** command in the UI to create a release manually from the the Releases or the Builds summary.
- By sending a command over the network to the [REST interface](#).

**However**, the action of creating a release **does not** mean it will automatically or immediately start a deployment. For example:

- There may be [deployment triggers](#) defined for an environment, which force the deployment to wait; this could be for a manual deployment, until a scheduled day and time, or for successful deployment to another environment.

- A deployment started manually from the **[Deploy]** command in the UI, or from a network command sent to the [REST interface](#), may specify a final target environment other than the last environment in a release pipeline. For example, it may specify that the release is deployed only as far as the QA environment and not to the production environment.
- There may be [queuing policies](#) defined for an environment, which specify which of multiple deployments will occur, or the order in which releases are deployed.
- There may be [pre-deployment approvers or gates](#) defined for an environment, and the deployment will not occur until all necessary approvals have been granted.
- Approvers may defer the release to an environment until a specified date and time using a [scheduled trigger](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Tasks for Build and Release

2/26/2018 • 5 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

A **task** is the building block for defining automation in a build definition, or in an environment of a release definition. A task is simply a packaged script or procedure that has been abstracted with a set of inputs. We provide some [built-in tasks](#) to enable fundamental build and deployment scenarios.

In addition, [Visual Studio Marketplace](#) offers a number of extensions; each of which, when installed to your account or collection, extends the task catalog with one or more tasks. Furthermore, you can write your own [custom extensions](#) to add tasks to your account in VSTS or your collection in TFS.

When you add a task to your build or release definition, it may also add a set of **demands** to the definition. The demands define the prerequisites that must be installed on the [agent](#) for the task to run. When you run the build or deployment, an agent that meets these demands will be chosen.

When you queue a build or a deployment, all the tasks are run in sequence, one after the other, on an agent. To run the same set of tasks in parallel on multiple agents, or to run some tasks without using an agent, see [phases](#).

## Task versions

Each task has a **Version** selector that enables you to specify the major version of the task used in your build or deployment. 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 definition and manually change to the new major version. The build or release log will include an alert that a new major version is available.

### Notes:

- If you select a preview version (such as **1.\* Preview**), be aware that this version is still under development and might have known issues.
- If you change the version and have problems with your builds, you can revert the definition change from the **History** tab. The ability to restore to an older version of a release definition is not currently available. You must manually revert the changes to the release definition, then save the definition.
- Consider cloning the definition and testing the cloned definition with the new major task version.

## Task control options

Each task offers you some **Control Options**.

### Enabled

Clear this check box to disable a task. This is useful when you want to temporarily take task out of the process for testing or for specific deployments.

**TIP**

You can also right-click the task to toggle this setting.

## Timeout

The timeout for this task in minutes. The default is zero (infinite timeout). Setting a value other than zero overrides the setting for the parent task phase.

## VSTS options

### Continue on error (partially successful)

Select this option if you want subsequent tasks in the same phase to possibly run even if this task fails. The build or deployment will be no better than partially successful. Whether subsequent tasks run depends on the **Run this task** setting.

### Run this task

Select the condition for running this task:

- Only when all previous tasks have succeeded
- Even if a previous task has failed, unless the build or release was canceled
- Even if a previous task has failed, even if the build was canceled
- Only when a previous task has failed
- [Custom conditions](#)

**NOTE**

If you're running tasks in cases when the build is canceled, then make sure you specify sufficient time for these tasks to run the [definition options](#).

## TFS 2015 and newer options

### Continue on error (partially successful)

Select this option if you want subsequent tasks in the same phase to run even if this task fails. The build or deployment will be no better than partially successful.

### Always run

Select this check box if you want the task to run even if the build or deployment is failing.

## Build tool installers (VSTS)

**VSTS preview feature**

To use this capability you must be working on VSTS and enable the [Task tool installers preview feature](#) for your account.

Tool installers enable your build process to install and control your dependencies. Specifically, you can:

- Install a tool or runtime on the fly (even on [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 process to run and validate your app for multiple versions of Node.js.

### Example: Test and validate your app on multiple versions of Node.js

## TIP

Want a visual walkthrough? See [our April 19 news release](#).

Create a new build definition (start with an empty process) to try this out.

### Tasks tab

Add these tasks:



Tool: Node Installer

- Version Spec:

```
$(NodeVersionSpec)
```



Utility: Command Line

- Tool (if you're running on a Windows agent)

```
where
```

- Tool (if you're running on a macOS or Linux agent)

```
which
```

- Arguments

```
node
```

### Variables tab

On the [Variables tab](#) define this variable:

NAME	VALUE	SETTABLE AT QUEUE TIME
NodeVersionSpec	6.x, 7.x	Selected

### Options tab

On the [Options tab](#):

1. Enable **Multi-configuration**.

2. Specify **Multipliers**:

```
NodeVersionSpec
```

3. (Optional) Select **Parallel** if you have multiple build agents and want to run your builds in parallel.

### Save & queue

Click **Save & queue**. Observe how two builds are run. The [Node Tool Installer](#) downloads each of the Node.js versions if they are not already on the agent. The [Command Line](#) task logs the location of the Node.js version on disk.

## Tool installer tasks

For a list of our tool installer tasks, see [Tool installer tasks](#).

## Related topics

- [Task phases](#)
- [Task groups](#)
- [Built-in task catalog](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Phases in Build and Release Management

2/28/2018 • 11 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017

## NOTE

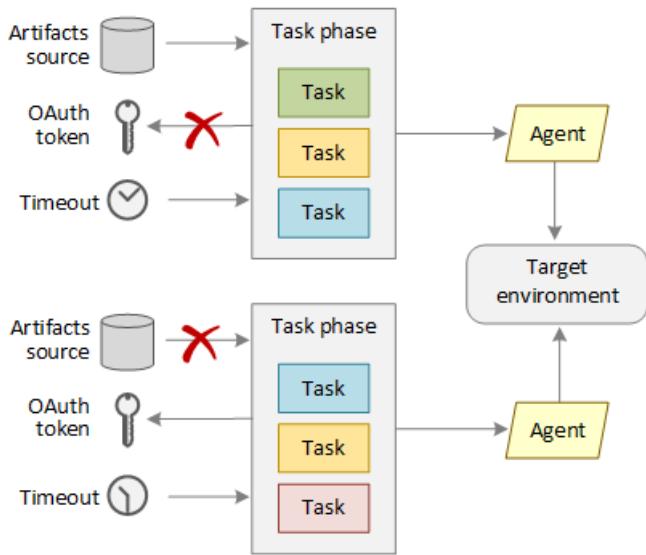
Some of the features described below are not yet available in Build. Some features are not yet available in Release Management. Wherever appropriate, each feature below is tagged to indicate whether it is presently available in Build or Release Management, and whether it is planned to be made available in one or the other.

Tasks can run in different **phases**, which effectively represent different execution locations.

By using different task phases in a build or release definition, you can:

ACTIVITY	RELEASE IN VSTS	RELEASE IN TFS 2017	RELEASE IN TFS 2018	BUILD IN VSTS
Partition your deployment process into sections that run on agents and sections that run without an agent	Yes	Yes	Yes	Yes
Partition your build or deployment process into sections where tasks in each section can target a different set of private agents using different demands	Yes	Yes	Yes	Yes
Partition your deployment process into sections where tasks in each section can target a different agent queue	Yes	Yes	Yes	Planned
Introduce a manual intervention task where deployment pauses while an operator carries out manual processes or validates the state of the system before continuing the deployment	Yes	Yes	Yes	No

ACTIVITY	RELEASE IN VSTS	RELEASE IN TFS 2017	RELEASE IN TFS 2018	BUILD IN VSTS
Switch off downloading of artifacts for sets of tasks that include their own logic for accessing source artifacts, or that do not require access to artifacts, which can reduce execution time and improve deployment efficiency	Yes	Yes	Yes	No
Switch off checking out code for sets of tasks that include their own logic for accessing sources, or that do not require sources, which can reduce build time	No	No	No	Planned
Publish build artifacts in one phase and consume those in subsequent phases	No	No	No	Yes
Run multiple phases in parallel	Planned	Planned	Planned	Planned
Permit access to OAuth tokens for only the tasks that need to interact with VSTS or TFS	Yes	Yes	Yes	Yes
Specify different execution timeouts for different sets of tasks to maximize deployment performance and control	Yes	Yes	Yes	Yes
Specify the conditions under which the tasks in the phase will execute; for example, when a previous phase has failed or when a <a href="#">custom condition</a> defined by an expression is true	Yes	No	Yes	Planned



The task phases you can use are:

- **Agent phase.** In this phase, tasks are executed on the computer(s) that host the build and release agent.
- **Agentless phase.** In this phase, tasks are orchestrated by and executed on VSTS or TFS. An agentless phase does not require an agent or any target computers.
- **Deployment group phase.** In this phase, tasks are executed on the computer(s) defined in a [deployment group](#), each of which hosts the build and release agent. This phase is only applicable to release definitions.

Screenshot of the VSTS Pipeline interface for the 'Fabrikam' project. The pipeline is set to 'Tasks' mode. The 'Dev' environment is selected, showing an error message: 'Environment 'Dev' should have atleast one phase.' A context menu is open at the top right of the environment card, with the '...' option highlighted. The menu options are:

- Add agent phase
- Add deployment group phase
- Add agentless phase
- Learn more about phases

By default, tasks you add to a build definition or release definition run in a single default agent phase. To add tasks to a specific phase when you have more than one phase in the definition, select the target phase and then choose +.

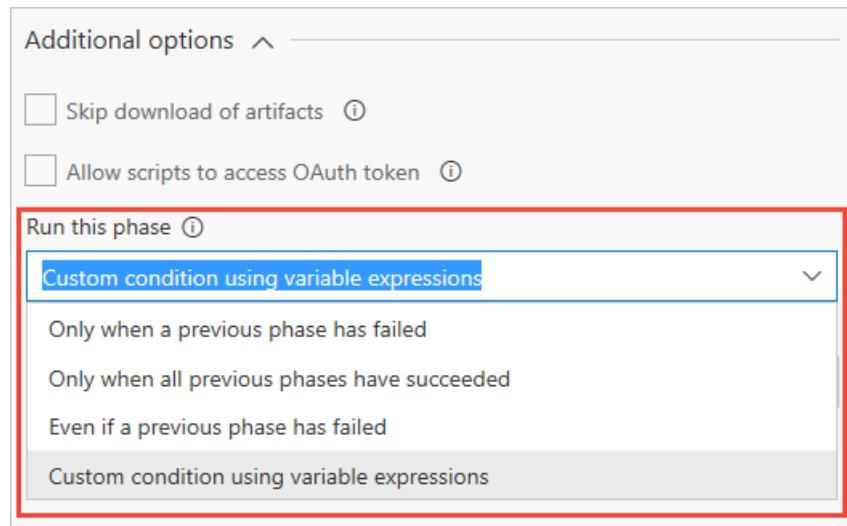
## Agent phase

An **agent phase** is a way of defining a sequence of tasks that will run on one or more agents. At run time, one or more jobs are created to be run on agents that match the demands specified in the phase properties.

The screenshot shows the 'Tasks' tab selected in the 'Fabrikam' release definition. A red box highlights the 'Run on agent' task item, which has a plus sign and three vertical dots icon. To the right, the 'Agent phase' configuration is shown, including fields for 'Display name' (set to 'Run on agent'), 'Agent selection', 'Queue' (set to 'Hosted'), and 'Demands'. Below these are sections for 'Execution plan' and 'Additional options' with checkboxes for skipping artifact download and allowing OAuth token access.

You can configure the following properties for an agent phase:

- **Display name:** The name displayed in the agent phase item in the task list.
- **Queue [Release (VSTS, TFS 2017 and newer), Build (Planned)]:** Use this option to specify the agent queue in which the jobs will run. In the case where multiple jobs are created, all the jobs run on agents within the same agent queue.
- **Demands:** Use these settings to specify how an agent for executing the tasks will be selected. In the case where multiple jobs are created, all the agents must have capabilities that satisfy the demands. For more details, see [Capabilities](#).
- **Skip download of artifacts [Release (VSTS, TFS 2017 and newer)]:** When used in a release definition, you may choose to skip the [download of artifacts](#) during the job execution. Use this option if you want to implement your own custom logic for downloading artifacts by using tasks, or if the tasks in a particular phase do not rely on the artifacts.
- **Allow scripts to access OAuth token [Release (VSTS, TFS 2017 and newer), Build (Planned)]:** Use this option if you want to allow tasks running in this phase access to the current VSTS or TFS OAuth security token. This is useful in many scenarios, such as when you need to run a custom PowerShell script that invokes the REST APIs on VSTS - perhaps to create a work item or query a build for information.
- **Run this phase [Release (VSTS, TFS 2018 Update 2 onwards), Build (Planned)]:** Use this option to run the tasks in the phase only when specific [conditions](#) are met. Select a predefined condition, or select "custom" and enter an [expression](#) that evaluates to **true** or **false**. Nested expressions can be used, and the expressions can access variables available in the release definition.



- **Timeout:** Use this option if you want to specify the timeout in minutes for jobs in this phase. 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](#).

### Parallel and multiple execution using agent phases

You can use multiple agents to run parallel jobs if you configure an agent phase to be **Multi-configuration** or **Multi-agent**.

Here are some examples where **multi-configuration** is appropriate:

#### NOTE

These options are available in Release (VSTS, TFS 2017 and newer) and Build (VSTS)

- **Multiple execution builds:** An agent phase can be used in a build definition to build multiple configurations in parallel. For example, you could build a C++ app for both debug and release configurations on both x86 and x64 platforms. To run multiple jobs, you identify a variable named a **multiplier**, and specify a list of values for that multiplier. A separate job is run for each value in the list. For example, you can run two parallel jobs if you define a variable named **BuildConfiguration** with the value `Debug,Release` and specify that variable to be used as a multiplier in the build definition. When you identify two variables to act as a multiplier, each with two values, you effectively create four jobs, one for each combination of values of the two variables. For more details and an example of multi-configuration build, see [Visual Studio Build](#).
- **Multiple execution deployments:** An agent phase can be used in an environment of a release definition to run multiple deployment jobs in parallel. For example, to configure your application to be load balanced across a site in US and a site in Europe, you can define a variable named **Location** with the value `US,Europe`, and specify that variable to be used as a multiplier in the agent phase of the environment. If two agents are available, both the jobs will be run simultaneously - one with **Location = US**, and another with **Location = Europe**. If only one agent is available, the first job is run, followed by the second job. When a multiplier results in many jobs being created in parallel, you can also specify the maximum number of agents that can be used for parallelism. For example, if **Location** has ten values, you can restrict to five sites being deployed in parallel by specifying **5** for the maximum number of agents.
- **Multiple execution testing:** An agent phase can be used in a build definition or in an environment of a release definition to run a set of tests in parallel - once for each test configuration. For example, to run a suite of web tests, once for each browser, you can define a variable named **Browser** with the value `IE,Chrome,Edge,Firefox` and specify that variable to be used as a multiplier. As the **agent demands** are specified only once for an agent phase, all agents should be pre-installed with all the browsers. As with

multiple executions deployment, you can specify the maximum number of agents that can be used at the same time for multi-configuration testing.

With multi-configuration you can run multiple jobs, each with a different value for one or more variables (multipliers). If you want to run the same job on multiple agents, then you can use **multi-agent** option of parallelism. Here is an example of when you need multi-agent parallelism.

- **Test slicing [Release (TFS2018, VSTS), Build (Planned)]:** An agent phase 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 multi-agent parallelism, the tasks in the phase 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. Select **Multi-agent** and specify the number of agents to use.

To use **Multipliers** for build or deployment, you must:

- Define one or more [variables](#) on the **Variables** tab of the definition or in a [variable group](#). Each variable, known in this context as a *multiplier* variable, must be defined as a comma-delimited list of the values you want to pass individually to the agents.
- Enter the name of the multiplier variable, without the \$ and parentheses, as the value of the **Multipliers** parameter.
- If you want to execute the phase for more than one multiplier variable, enter the variable names as a comma-delimited list - omitting the \$ and parentheses for each one.
- If you want to limit the number of agents used during the deployment to a number less than you have configured for your account, enter that value as the **Maximum number of agents** parameter.

For example, you might define two variables named **Location** and **Browser** as follows::

- **Location** = `US,Europe`
- **Browser** = `IE,Chrome,Edge,Firefox`

The following configuration will execute the deployment eight times using a maximum of four agents at any one time:

- **Multipliers** = `Location,Browser`
- **Maximum number of agents** = `4`

## Agentless phase

Use an agentless phase in a build or release definition to run tasks that do not require an agent, and execute entirely on the VSTS or TFS. Only a few tasks, such as the [Manual Intervention](#) and [Invoke REST API](#) tasks, are supported in an agentless phase at present. The properties of an agentless phase are similar to those of the [agent phase](#).

Fabrikam

Save Release View releases ...

Pipeline Tasks Variables Retention Options History

Dev Deployment process

Run on agent Run on agent

Deploy Azure App Service Azure App Service Deploy

Quick Web Performance... Cloud-based Web Performance Test

Agentless phase Agentless phase

Display name \* Agentless phase

Execution plan

Parallelism Single execution Multiple executions

Multipliers

Continue on error

Timeout \* 0

Additional options

The screenshot shows the 'Tasks' tab of a pipeline definition. A specific phase, 'Agentless phase', is highlighted with a red box. This phase is set to run on a server ('Run on server') and has a display name of 'Agentless phase'. It includes an 'Execution plan' section with parallelism settings ('Single execution' is selected), multipliers, and a checkbox for 'Continue on error'. The 'Timeout' is set to 0. The 'Additional options' section is collapsed.

## Deployment group phase

### NOTE

Deployment group phases can only be used in release definitions. They cannot be used in build definitions.

Deployment groups make it easy to define groups of target servers for deployment, and install the required agent on each one. Tasks that you define in a deployment group phase run on some or all of the target servers, depending on the arguments you specify for the tasks and the phase 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. For more details, see [Deployment groups](#). You can also use the slider control to specify the proportion of the target servers that the process 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.

The screenshot shows the 'Fabrikam' pipeline configuration in the Azure DevOps interface. The 'Tasks' tab is selected. A 'Deployment group phase' card is open, with its title 'Deployment group phase' highlighted by a red box. The card contains fields for 'Display name' (set to 'Deployment group phase'), 'Deployment targets' (set to 'FabrikamGroup'), 'Required tags' (empty), 'Targets to deploy to in parallel' (set to 'Multiple'), 'Maximum number of targets in parallel' (set to 100%), 'Timeout' (set to 0), and two optional checkboxes for 'Skip download of artifacts' and 'Allow scripts to access OAuth token'. The background shows other pipeline components like 'Dev Deployment process' and task cards for 'Azure Deployment' and 'Azure PowerShell'.

The timeout and additional options of a deployment group phase are the same as those of the [agent phase](#).

## Multiple phases

You can add multiple phases to a build or release definition, and then add tasks to each one by selecting the target phase for the new tasks.

Multiple phases can only be used in Release Management in VSTS and TFS 2017 and newer, and in Build in VSTS.

For example, the definition shown below divides the overall release execution into separate execution phases by using two agent phases and an agentless phase.

The screenshot shows the VSTS Release Pipeline interface for a project named "Fabrikam". The pipeline is currently at the "Dev" stage. On the left, there's a list of tasks: "Agent phase" (Azure PowerShell script), "File Copy", "Agentless phase" (Manual Intervention), another "Agent phase" (Azure App Service Deploy), and a "Control Options" section. On the right, the "Manual Intervention" task is being configured. It has a "Display name" of "Manual Intervention" and instructions to "Validate deployment conditions for all external systems before continuing with deployment". Under "Notify users", "Mateo Escobedo" is selected. In the "On timeout" section, the "Reject" option is chosen. The "Control Options" section includes a "Control Options" dropdown.

In the example above:

1. The tasks in the first phase of the release run on an agent and, after this phase is complete, the agent is released.
2. The agentless phase contains a Manual Intervention task that runs on the VSTS or TFS. It does not execute on, or require, an agent or any target servers. The Manual Intervention task displays its message and waits for a "resume" or "reject" response from the user. In this example, if the configured timeout is reached, the task will automatically reject the deployment (set the timeout in the control options section to zero if you do not want an automated response to be generated).
3. If the release is resumed, tasks in the third phase run - possibly on a different agent. If the release is rejected, this phase does not run and the release is marked as failed.

It's important to understand some of the consequences of phased execution:

- Each phase may use different agents. You should not assume that the state from an earlier phase is available during subsequent phases.
- The **Continue on Error** and **Always run** options for tasks in each phase do not have any effect on tasks in subsequent phases of the build or release. For example, setting **Always run** on a task at the end of the first phase will not guarantee that tasks in subsequent phases will run.

## Related topics

- [Tasks](#)
- [Task groups](#)
- [Specify conditions for running a task](#)

# Specify conditions for running a task

9/12/2017 • 6 min to read • [Edit Online](#)

## VSTS

Inside the **Control Options** of each task, and in the **Additional options** for a phase in a release definition, you can specify the conditions under which the task or phase will run:

- Only when all previous tasks have succeeded
- Even if a previous task has failed, unless the build or release was canceled
- Even if a previous task has failed, even if the build was canceled
- Only when a previous task has failed
- Custom conditions

## Enable a custom condition

If the built-in conditions don't meet your needs, then you can specify **custom conditions**.

Express the condition as a nested set of functions. The agent evaluates the innermost function and works its way out. The final result is a boolean value that determines if the task is run or not. Details on syntax are described below.

Do any of your conditions make it possible for the task to run even after the build is canceled by a user? If so, then specify a reasonable value for **Build job cancel timeout in minutes** in the [options](#) so that these kinds of tasks have enough time to complete after the user clicks **Cancel**.

## Examples

### Run for the master branch, if succeeding

```
and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/master'))
```

### Run if the branch is not master, if succeeding

```
and(succeeded(), ne(variables['Build.SourceBranch'], 'refs/heads/master'))
```

### Run for user topic branches, if succeeding

```
and(succeeded(), startsWith(variables['Build.SourceBranch'], 'refs/heads/users/'))
```

### Run for continuous integration (CI) builds if succeeding

```
and(succeeded(), in(variables['Build.Reason'], 'IndividualCI', 'BatchedCI'))
```

### Run if the build is run by a branch policy for a pull request, if failing

```
and(failed(), eq(variables['Build.Reason'], 'PullRequest'))
```

### Run if the build is scheduled, even if failing, even if canceled

```
and(always(), eq(variables['Build.Reason'], 'Schedule'))
```

**Release.Artifacts.{artifact-alias}.SourceBranch** is equivalent to **Build.SourceBranch**.

## Types

### Boolean

`true` or `false` (ordinal case insensitive)

### Null

Null is a special type that is returned from a dictionary miss only, e.g. (`variables['noSuch']`). There is no keyword for null, but you can test for it by using the implicit type casting described below.

### Number

Starts with `-` `.` or `0-9`. Cannot contain `,`.

### String

Must be single-quoted. For example: `'this is a string'`.

To express a literal single-quote, escape it with a single quote. For example:

`'It''s OK if they're using contractions.'`.

### Version

A version number with up to four segments. Must start with a number and contain two or three period (`.`) characters. For example: `1.2.3.4`.

## Type Casting

### Conversion chart

Detailed conversion rules are listed further below.

		TO				
		Boolean	Null	Number	String	Version
From	Boolean	-	-	Yes	Yes	-
	Null	Yes	-	Yes	Yes	-
	Number	Yes	-	-	Yes	Partial
	String	Yes	Partial	Partial	-	Partial
	Version	Yes	-	-	Yes	-

### Boolean to Number

False => 0

True => 1

### **Boolean to String**

False => 'False'

True => 'True'

### **Null to Boolean**

=> False

### **Null to Number**

=> 0

### **Null to String**

=> Empty string

### **Number to Boolean**

0 => False

Otherwise => True

### **Number to Version**

Must be greater than zero and must contain a non-zero decimal. Must be less than [Int32.MaxValue](#) (decimal component also).

### **Number to String**

Converts the number to a string with no thousands separator and no decimal separator.

### **String to Boolean**

Empty string => False

Otherwise => True

### **String to Null**

Empty string => Null

Otherwise not convertible

### **String to Number**

Empty string => 0

Otherwise try-parse using [InvariantCulture](#) and the following rules: AllowDecimalPoint | AllowLeadingSign | AllowLeadingWhite | AllowThousands | AllowTrailingWhite. If try-parse fails, then not convertible.

### **String to Version**

Try-parse. Must contain Major and Minor component at minimum. If try-parse fails, then not convertible.

### **Version to Boolean**

=> True

### **Version to String**

Major.Minor or Major.Minor.Build or Major.Minor.Build.Revision.

## **Variables**

Alias to reference a build variable. For example:

- Index syntax: `variables['Build.SourceBranch']`

- Property dereference syntax: `variables.Build.SourceBranch`. In order to use this syntax, the property name must:
  - Start with `a-z` or `_`
  - Be followed by `a-z` `0-9` or `_`

Some of the more useful predefined variables include:

- `Build.Reason` which you can use to check whether the build was the result of a [build trigger](#), a [Git PR affected by a branch policy](#), or a [TFVC gated check-in](#).
- `Build.SourceBranch`
- `Release.Artifacts.{artifact-alias}.SourceBranch`

For details on these and other variables, including predefined variables and their possible values, see [Build variables](#) and [Release variables](#).

## Job status functions

### **always**

- Always evaluates True (even when canceled). Note: A critical failure may still prevent a task from running. For example, if getting sources failed.
- Min parameters: 0. Max parameters: 0

### **canceled**

- Evaluates True when `eq(variables['Agent.JobStatus'], 'Canceled')`.
- Min parameters: 0. Max parameters: 0

### **failed**

- Evaluates True when `eq(variables['Agent.JobStatus'], 'Failed')`.
- Min parameters: 0. Max parameters: 0

### **succeeded**

- Evaluates True when `in(variables['Agent.JobStatus'], 'Succeeded', 'PartiallySucceeded')`
- Min parameters: 0. Max parameters: 0

### **succeededOrFailed**

- Evaluates True when `in(variables['Agent.JobStatus'], 'Succeeded', 'PartiallySucceeded', 'Failed')`
- Min parameters: 0. Max parameters: 0

## General functions

### **and**

- Evaluates True if all parameters are True
- Min parameters: 2. Max parameters: N
- Casts parameters to Boolean for evaluation
- Short-circuits after first False

### **contains**

- Evaluates True if left parameter String contains right parameter
- Min parameters: 2. Max parameters: 2
- Casts parameters to String for evaluation

- Performs ordinal ignore-case comparison

### **endsWith**

- Evaluates True if left parameter String ends with right parameter
- Min parameters: 2. Max parameters: 2
- Casts parameters to String for evaluation
- Performs ordinal ignore-case comparison

### **eq**

- Evaluates True if parameters are equal
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Returns False if conversion fails.
- Ordinal ignore-case comparison for Strings

### **ge**

- Evaluates True if left parameter is greater than or equal to the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

### **gt**

- Evaluates True if left parameter is greater than the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

### **in**

- Evaluates True if left parameter is equal to any right parameter
- Min parameters: 1. Max parameters: N
- Converts right parameters to match type of left parameter. Equality comparison evaluates False if conversion fails.
- Ordinal ignore-case comparison for Strings
- Short-circuits after first match

### **le**

- Evaluates True if left parameter is less than or equal to the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

### **lt**

- Evaluates True if left parameter is less than the right parameter
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Errors if conversion fails.
- Ordinal ignore-case comparison for Strings

### **ne**

- Evaluates True if parameters are not equal
- Min parameters: 2. Max parameters: 2
- Converts right parameter to match type of left parameter. Returns True if conversion fails.
- Ordinal ignore-case comparison for Strings

## not

- Evaluates True if parameter is False
- Min parameters: 1. Max parameters: 1
- Converts value to Boolean for evaluation

## notIn

- Evaluates True if left parameter is not equal to any right parameter
- Min parameters: 1. Max parameters: N
- Converts right parameters to match type of left parameter. Equality comparison evaluates False if conversion fails.
- Ordinal ignore-case comparison for Strings
- Short-circuits after first match

## or

- Evaluates True if any parameter is true
- Min parameters: 2. Max parameters: N
- Casts parameters to Boolean for evaluation
- Short-circuits after first True

## startsWith

- Evaluates true if left parameter string starts with right parameter
- Min parameters: 2. Max parameters: 2
- Casts parameters to String for evaluation
- Performs ordinal ignore-case comparison

## xor

- Evaluates True if exactly one parameter is True
- Min parameters: 2. Max parameters: 2
- Casts parameters to Boolean for evaluation

## Q&A

### What about string parsing and other operations?

We might add these later. [Vote on user voice](#)

### I've got a condition that runs even when build was cancelled. Does this affect a build that I cancelled in the queue?

No. If you cancel a build while it's in the queue, then the entire build is canceled, including tasks like this.

### I've got a task condition that runs even when the deployment was canceled. How do I specify this?

This scenario is not yet supported for release definitions.

# Concurrent release pipelines in Team Foundation Server

12/19/2017 • 5 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017**

This article describes the licensing model for Release Management in Team Foundation Server 2017 (TFS 2017) or later. We don't charge you for Team Foundation Build (TFBuild) so long as you have a TFS Client Access License (CAL).

A TFS *concurrent pipeline* gives you the ability to run a single release at a time in a team project collection. You can keep hundreds or even thousands of release definitions in your collection. But, to run more than one release at a time, you need additional concurrent pipelines.

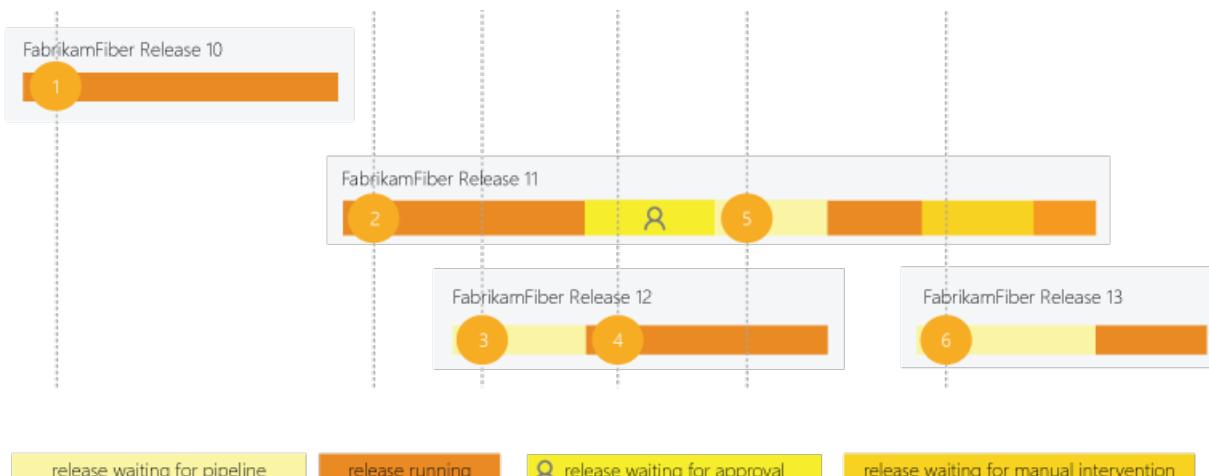
One free concurrent pipeline is included with every collection in a Team Foundation server. Every Visual Studio Enterprise subscriber in a Team Foundation server contributes one additional concurrent pipeline. You can buy additional private pipelines from the Visual Studio Marketplace.

Do I need concurrent pipelines in TFS 2015 or TFS 2013? Short answer: no. [More details](#)

## How a concurrent pipeline is consumed

For example, a collection in a Team Foundation server has one concurrent pipeline. 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 concurrent pipeline only when it is being actively deployed to an environment. Waiting for an approval does not consume a concurrent pipeline. However, waiting for a manual intervention in the middle of a deployment does consume a concurrent pipeline.



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 concurrent pipeline.
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

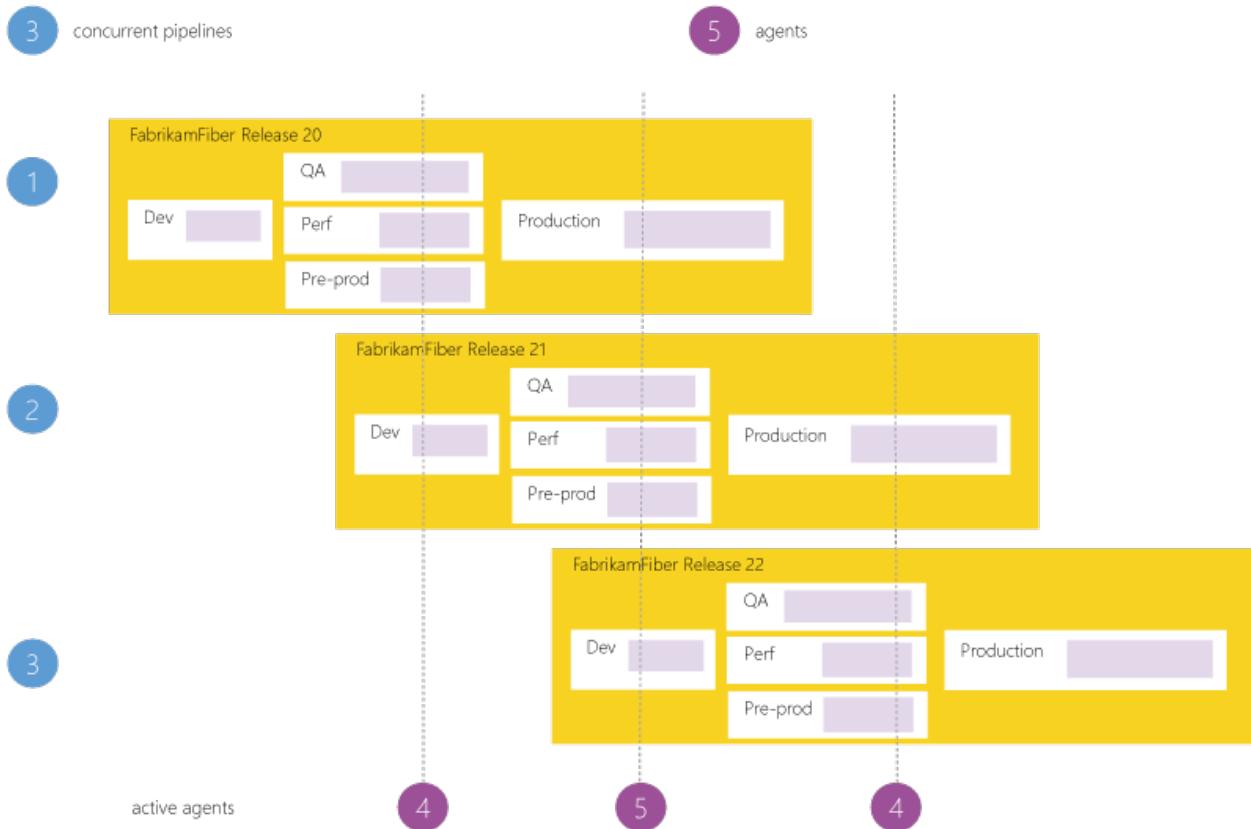
consumes a concurrent pipeline.

Manual intervention does not consume a pipeline in TFS 2017.1 and newer.

## Concurrent processing within a single release

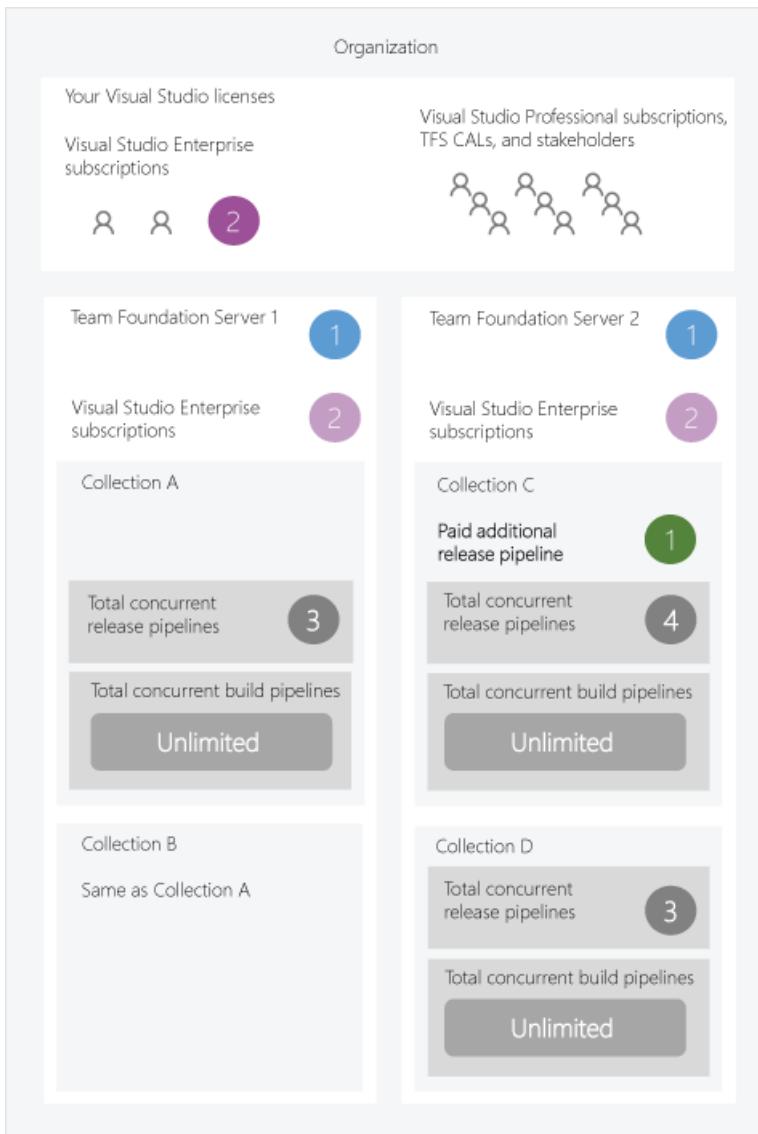
Concurrent processing within a single release does not require additional concurrent pipelines. So long as you have enough agents, you can deploy to multiple environments in a release at the same time.

For example, suppose your collection has three concurrent pipelines. 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 concurrent pipelines.



## Concurrent pipelines in an organization

For example, here's an organization that has multiple Team Foundation Servers. Two of their users have Visual Studio Enterprise subscriptions that they can use at the same time across all their on-premises servers and in each collection so long as the customer adds them as users to both the servers as explained below.



## Determine how many concurrent pipelines you need

You can begin by seeing if your teams can get by with the concurrent pipelines you've got by default. As the number of queued releases exceeds the number of concurrent pipelines you have, your release queues will grow longer. When you find the queue delays are too long, you can purchase additional concurrent pipelines as needed.

### Simple estimate

A simple rule of thumb: Estimate that you'll need one concurrent pipeline for every 10 users in your server.

### Detailed estimate

In the following scenarios you might need multiple concurrent pipelines:

- 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 concurrent pipeline for each team.
- If you develop multiple applications in one collection, then you'll likely need additional concurrent pipelines: 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 concurrent pipeline 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, the text "Team Foundation Server", a dropdown arrow, and links for "Home", "Rooms", and a gear icon. Below this, a secondary navigation bar has three items: "Control panel", "Access levels" (which is underlined in blue, indicating it's the active page), and "Legacy extensions".

URL example: `http://{your_server}:8080/tfs/_admin/_licenses`

2. On the left side of the page, click **VS Enterprise**.
3. Add your users who have Visual Studio Enterprise subscriptions.

After you've added these users, additional licenses will appear on the resource limits page described below.

## Purchase additional concurrent pipelines

If you need to run more concurrent releases, you can [buy additional private pipelines from the Visual Studio marketplace](#). Since there is no way to directly purchase concurrent pipelines from Marketplace for a TFS instance at present, you must first buy concurrent pipelines for a VSTS account. After you buy the private pipelines for a VSTS account, you enter the number of purchased concurrent pipelines manually on the resource limits page described below.

## View and manage concurrent pipelines

1. Browse to **Collection settings, Build and Release, Resource limits**.

The screenshot shows the "Build and Release" section of the collection settings. The top navigation bar includes the TFS logo, "DefaultCollection", a dropdown arrow, and links for "Home", "Rooms", and a gear icon. Below this, a secondary navigation bar has tabs for "Overview", "Users", "Security", "Build and Release" (which is underlined in blue), "Agent pools", and "Extensions". At the bottom of this bar are "Settings" and "Resource limits".

URL example: `http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue?_a=resourceLimits`

2. View or edit the number of purchased concurrent pipelines.

## Q&A

### Who can use the system?

TFS users with a [TFS CAL](#) can author as many releases as they want.

To approve releases, a TFS CAL is not necessary; any user with [stakeholder access](#) can approve or reject releases.

### Do I need concurrent pipelines to run builds on TFS?

No, on TFS you don't need concurrent pipelines to run builds. You can run as many builds as you want at the same time for no additional charge.

### Do I need concurrent pipelines to use release management in versions before TFS 2017?

No.

In TFS 2015, so long as your users have a TFS CAL, they can use release management for no additional charge in trial mode. We called it "trial mode" to indicate that we would eventually charge for release management. Despite this label, we fully support release management in TFS 2015.

In TFS 2013 there is no change in the licensing of release management features.

### How is release management licensed in VSTS?

See [concurrent pipelines in VSTS](#).

# Concurrent build and release pipelines in VSTS

12/19/2017 • 5 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#)

## NOTE

October 2017 update: We're temporarily providing unlimited private pipelines while we work on fixing an issue. We estimate that we'll fix the issue and return to providing only the private pipelines you have sometime in November 2017.

A VSTS *concurrent pipeline* gives you the ability to run a single build or a single release at a time in your account. There are two types of concurrent pipelines in VSTS.

## Private pipelines

If you want to run builds and releases on your own machines ([private agents](#)), then you need *private pipelines*. We provide one free private pipeline in each VSTS account. In addition, every active Visual Studio Enterprise subscriber in your account contributes a free private pipeline. You can buy [additional private pipelines](#) from the Visual Studio Marketplace. After you've done this, you can deploy your own [private agents](#) and use them with these private pipelines. You can register any number of private agents with your account for no additional charge.

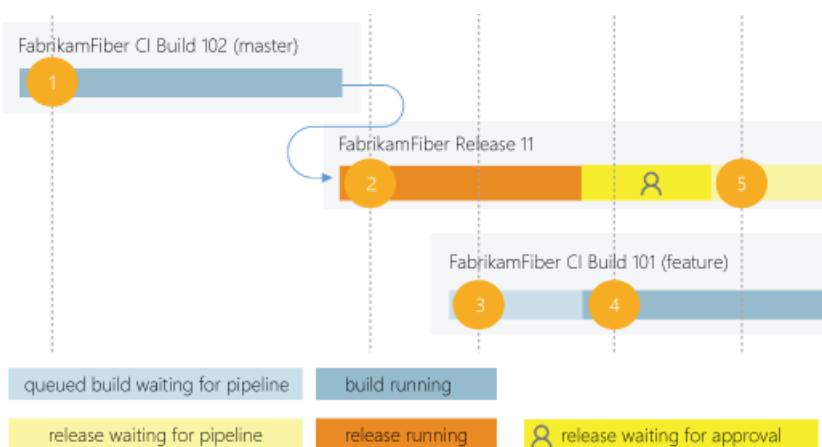
## Hosted pipelines

If you want to run builds and releases on machines managed by Microsoft ([hosted agents](#)), then you need hosted pipelines. We provide 240 minutes of free total compute time per month from a [hosted agent](#) to run a build or a release. Each build or release job within this free allocation cannot run for more than 30 minutes. To run more builds or releases concurrently, you can buy [additional hosted pipelines](#) from the Visual Studio Marketplace. With the first purchase of a hosted pipeline, the 240 minute limit on total build and release time as well as the 30 minute limit on a single job are waived. Each additional purchase of a hosted pipeline adds another hosted agent for running your builds and releases.

## How a concurrent pipeline is consumed

For example, consider a VSTS account that has only one concurrent pipeline. This allows users in that account to run only one build or release at a time. When additional builds and releases are triggered, they are queued and will wait for the previous one to complete.

A release requires a concurrent pipeline only when it is being actively deployed to an environment. Waiting for an approval or a manual intervention does not consume a concurrent pipeline.



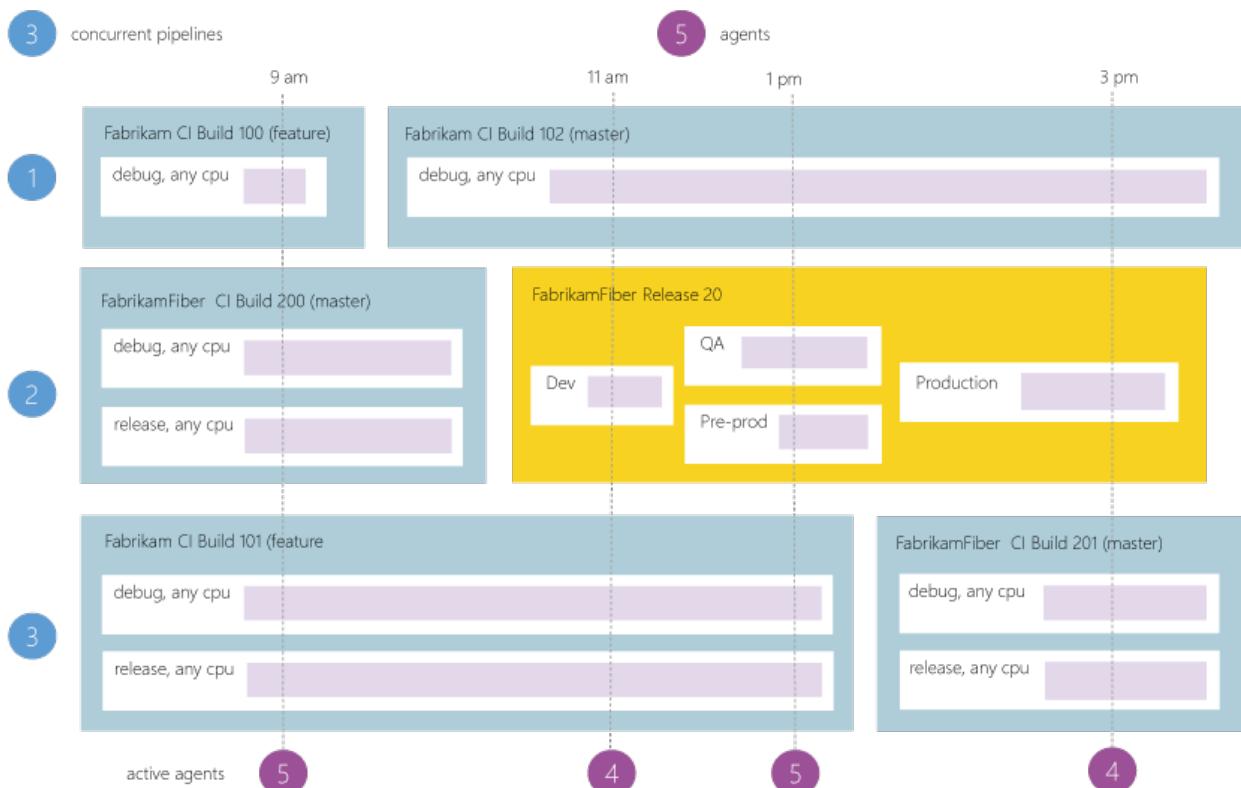
1. FabrikamFiber CI Build 102 (master branch) is first to be started.
2. Deployment of FabrikamFiber Release 11 is triggered by completion of FabrikamFiber CI Build 102.
3. FabrikamFiber CI Build 101 (feature branch) is triggered. The build can't start yet because Release 11's deployment is active. So the build stays queued.
4. Release 11 waits for approvals. Fabrikam CI Build 101 starts because a release waiting for approvals does not consume a concurrent pipeline.
5. Release 11 is approved. It resumes only after Fabrikam CI Build 101 is completed.

## Concurrent processing within a single build or release

Concurrent processing within a single build or release does not require additional concurrent pipelines. So long as you have enough agents, you can

- In a build process, run multiple build configurations at the same time.
- In a release process, deploy to multiple environments at the same time.

For example, suppose your VSTS account has three concurrent pipelines. You can have more than three agents running at the same time to perform parallel operations within builds and releases. For instance, notice below at 9 a.m. that five agents are actively running jobs from three concurrent pipelines.



## Determine how many concurrent pipelines you need

You can begin by seeing if your teams can get by with the concurrent pipelines you've got by default. As the number of queued builds and releases exceeds the number of concurrent pipelines you have, your build and release queues will grow longer. When you find the queue delays are too long, you can purchase additional concurrent pipelines as needed from Visual Studio Marketplace.

### Simple estimate

A simple rule of thumb: Estimate that you'll need one concurrent pipeline for every 10 users in your account.

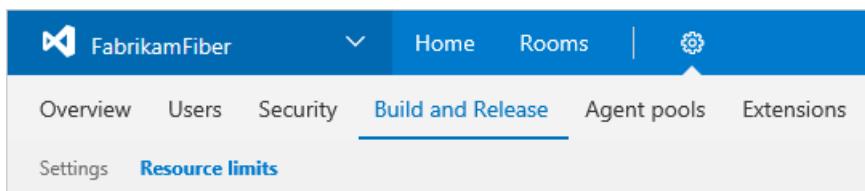
### Detailed estimate

In the following scenarios you might need multiple concurrent pipelines:

- If you have multiple teams, and if each of them require a CI build, then you'll likely need a concurrent pipeline for each team.
- If your CI build trigger applies to multiple branches, then you'll likely need a concurrent pipeline for each branch.
- If you develop multiple applications using one account or server, then you'll likely need additional concurrent pipelines: one to deploy each application at the same time.

## View available pipelines

1. Browse to **Account settings, Build and Release, Resource limits**.



URL example: `https://{{your_account}}/_admin/_buildQueue?a=resourceLimits`

2. View the maximum number of concurrent pipelines that are available in your account.
3. Select **Pipelines queue...** to display all the builds and releases that are actively consuming an available pipeline or that are queued waiting for a pipeline to be available.

## Sharing of pipelines across projects in a collection

Pipelines are purchased at the account level, and they are shared amongst all projects in an account. We don't yet offer a way to partition or dedicate certain pipelines to a specific project or agent pool. For example:

1. You purchase two pipelines in your account.
2. You queue two builds in the first project, and both the pipelines are consumed.
3. You queue a build in the second project. That build will not start until one of the builds in your first project is completed.

In the future, we plan to support finer control on allocation of pipelines.

## Q&A

### Who can use the Build and Release Management features?

VSTS users with [basic access](#) can author as many builds and releases as they want.

To approve releases, basic access is not necessary. Any user with [stakeholder access](#) can approve or reject releases.

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

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

**I use XAML build controllers with my account. How am I charged for those?**

You can register one XAML build controller for each private pipeline in your account. Your account gets at least one free private pipeline, so you can register one XAML build controller for no additional charge. For each additional XAML build controller, you'll need an additional private pipeline.

**As a Visual Studio Enterprise subscriber, do I get additional pipelines for TFS and VSTS?**

Yes. Visual Studio Enterprise subscribers get [one concurrent private pipeline in Team Foundation Server 2017 or later](#) and one concurrent private pipeline in each VSTS account of which they are a member.

# Library

12/19/2017 • 1 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017

*Library* is a collection of *shared* build and release assets for a team project. Assets defined in a library can be used in multiple build and release definitions of the team project. The **Library** tab can be accessed directly in the **Build & Release** hub in Visual Studio Team Services (VSTS) 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 definitions in VSTS and TFS 2017 and newer at present. Task groups and service endpoints are available to build and release definitions in TFS 2015 and newer, and VSTS.

## 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 definitions. For example, you must be a 'User' for a variable group to be able to use it in a release definition.
Administrator	In addition to all the above operations, members of this role can manage membership of all other roles for the item. The user that created an item is automatically added to the Administrator role for that item.

The security settings for the **Library** tab control access for *all* items in the library. Role memberships for individual items are automatically inherited from those of the **Library** node. In addition to the three roles listed above, the **Creator** role on the library defines who can create new items in the library, but cannot be used to manage permissions for other users. By default, the following groups are added to the **Administrator** role of the library: **Build Administrators**, **Release Administrators**, and **Project Administrators**.

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Variable groups

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

## VSTS | TFS 2018 | TFS 2017

Use a variable group to store values that you want to make available across multiple build and release definitions. Variable groups are defined and managed in the **Library** tab of the **Build & Release** hub.

## Create a variable group

1. Open the **Library** tab to see a list of existing variable groups for your project. Choose **+ Variable group**.

The screenshot shows the VSTS interface with the 'Build and Release' hub selected. The 'Library' tab is highlighted with a red box. Below it, there's a search bar labeled 'Search variable groups' and a blue button labeled '+ Variable group' with a red box around it. A table below lists existing variable groups, with one row shown: 'HO Website Configu...' by 'Alex Homer' with a description 'Variables used to co...' and a date '4/26/2017'. The 'Variable groups' tab is underlined, indicating it's the active section.

Name ↑	Modified by	Description	Date modified
HO Website Configu...	Alex Homer	Variables used to co...	4/26/2017

2. Enter a name and description for the group. Then enter the name and value for each **variable** you want to include in the group, choosing **+ Add** for each one. If you want to encrypt and securely store the value, choose the "lock" icon at the end of the row. When you're finished adding variables, choose **Save**.

**Properties**

Variable group name

HO Website Configuration

Description

Variables used to configure head office website

 Link secrets from an Azure key vault as variables
**Variables**

Name ↑	Value	🔒
app-location	Head_Office	🔓
app-name	Fabrikam	
password	*****	
requires-login	True	
user-name	*****	

+ Add

Variable groups follow the [library security model](#).

## Link secrets from an Azure key vault as variables

Link an existing Azure key vault to a variable group and map selective vault secrets to the variable group.

1. In the **Variable groups** page, enable **Link secrets from an Azure key vault as variables**. You'll need an existing key vault containing your secrets. You can create a key vault using the [Azure portal](#).

**Link secrets from an Azure key vault as variables**

Azure subscription \* | [Manage](#)

This setting is required.

Key vault name \* [Manage](#)

This setting is required.

2. Specify your Azure subscription end point and the name of the vault containing your secrets.

Ensure the Azure endpoint has at least **Get** and **List** management permissions on the vault for secrets. You can enable VSTS 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 definition linked to the variable group during the build or release.
- Any changes made to *existing* secrets in the key vault, such as a change in the value of a secret, will be made available automatically to all the definitions in which the variable group is used.
- When *new* secrets are added to the vault, they are **not** made available automatically to all the definitions. New secrets must be explicitly added to the variable group in order to make them available to definitions in which the variable group is used.
- Azure Key Vault supports storing and managing cryptographic keys and secrets in Azure. Currently, VSTS variable group integration supports mapping only secrets from the Azure key vault. Cryptographic keys and certificates are not yet supported

## Use a variable group

To use a variable group, open the definition, select the **Variables** tab, select **Variable groups**, and then choose **Link variable group**.

The screenshot shows the 'Variables' tab of a VSTS pipeline definition for 'Fabrikam'. On the left, there's a sidebar with 'Process variables' and 'Variable groups' (which is highlighted with a red box). Below that is 'Predefined variables'. On the right, under 'Variable groups', it says 'There aren't any variable groups linked to this definition.' A 'Link variable group' button is highlighted with a red box. A modal window titled 'Link variable group' is open, showing a list of variable groups: 'HO Website Configuration (5)' (selected) and 'Dev' (selected). There are checkboxes for 'Production' and 'QA'. At the bottom of the modal is a 'Link' button.

You can link a variable group to a release definition, or to a specific environment in a release definition. When you link to a release definition, all the variables in the group are available for use in all environments of that definition. When you link to an environment, the variables from the variable group scoped to that environment and are not accessible in the other environments of the same release.

You access the value of the variables in a linked variable group in exactly the same way as [variables you define within the definition itself](#). For example, to access the value of a variable named **customer** in a variable group

linked to the definition, 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.

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 environments to which the variable group is linked.

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Task Groups

1/23/2018 • 3 min to read • [Edit Online](#)

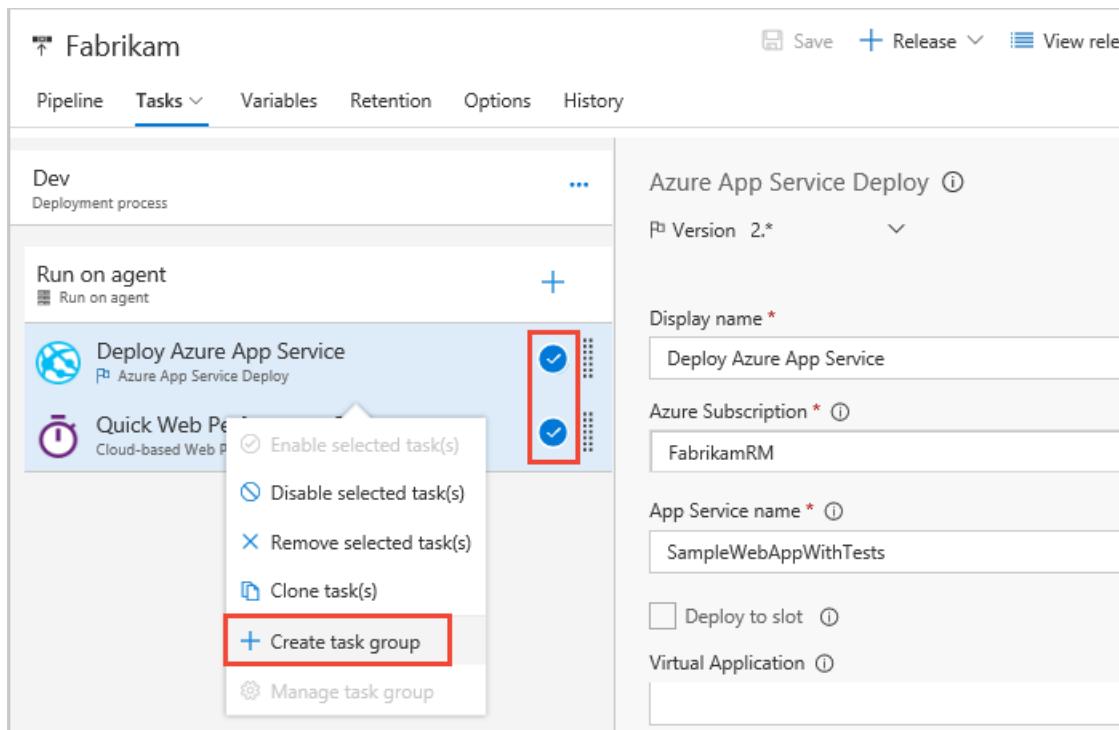
## VSTS | TFS 2018 | TFS 2017

A *task group* lets you to encapsulate a sequence of tasks already defined in a build or a release definition into a single reusable task that can be added to a build or release definition, 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 add to other release and build definitions. Task groups are stored at 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.

To create a task group, selecting a sequence of tasks in a build or release definition (when using a mouse, click on the checkmarks of each one). Then open the shortcut menu and choose **Create task group**. Specify a name and description, and the category (tab) you want to add it to in the Add tasks dialog.



However, before you do that, consider the following pointers to help you achieve the desired behavior:

- 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 definition.

- 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 deployment, 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.

All the task groups you create in the current project are listed in the **Task groups** tab of the **Build and Release** hub.

The screenshot shows the Azure DevOps interface for managing task groups. The top navigation bar includes 'Fabrikam' (selected), 'Home', 'Code', 'Work', 'Build & Release' (selected), 'Test', and a gear icon. The 'Build & Release' tab is expanded, showing 'Builds', 'Releases', 'Library', 'Task Groups' (selected and highlighted with a red box), and 'Deployment Groups'. On the left, a sidebar lists 'Task Groups' under 'Default', 'Deploy and Test', and 'FabrikamTests' (selected and highlighted with a blue box). The main content area shows 'FabrikamTests' details: Name: FabrikamTests, Description: Task group for Fabrikam tests, Category: Test. Below this is a 'Parameters' section with three entries: TestMachines (my-vm.corp.com, Default Value), TestPath, and PSscript.

- In the **Properties** page you can edit details of the task group, and change the default values and descriptions for the parameters.
- 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.
- 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 release definitions, build definitions, and other task groups that use (reference) this task group. This is useful to ensure changes do not have unexpected effects on other processes.

FabrikamTests

Properties Tasks History References **References**

Save

Expand each category for task group reference.

**Release definitions**

Name	Environments
Current Billing	Gallery , TFS
AccountMigration	Account Migration
Rotate Secrets	Deploy, Rotate Secrets , TFS

**Build definitions**

No references found.

**Task groups**

No references found.

You can import and export a task group as a JSON file.

Task Group: Config Change

Properties Tasks History References

Save

Name

Description

## Related topics

- [Tasks](#)
- [Task phases](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Service endpoints for Build and Release

1/19/2018 • 17 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

You will typically need to connect to external and remote services to execute tasks for a build or deployment. For example, you may need to connect to your Microsoft Azure subscription, to a different build server or file server, to an online continuous integration environment, or to services you install on remote computers.

Watch this video on Channel 9 to learn about service endpoints.

<https://channel9.msdn.com/Series/DevOps-Release-Management/Service-Endpoints-with-Visual-Studio-Team-Services/player>

You can define endpoints in Visual Studio Team Services (VSTS) or Team Foundation Server (TFS) that are available for use in all your tasks. For example, you can create an endpoint for your Azure subscription and use this endpoint name in an Azure Web Site Deployment task in a release definition.

You define and manage service endpoints from the Admin settings of your team project.

- VSTS: [https://{{account}}.visualstudio.com/{{teamproject}}/\\_admin/\\_services](https://{{account}}.visualstudio.com/{{teamproject}}/_admin/_services)
- TFS: [https://{{tfsserver}}/{{collection}}/{{teamproject}}/\\_admin/\\_services](https://{{tfsserver}}/{{collection}}/{{teamproject}}/_admin/_services)

Service endpoints are created at project scope. An endpoint created in one project is not visible in another team project.

## Common endpoint types

VSTS and TFS support a variety of endpoint types by default. Some of these are described below:

- [Azure Classic service endpoint](#)
- [Azure Resource Manager service endpoint](#)
- [Azure Service Bus service endpoint](#)
- [Azure Service Fabric service endpoint](#)
- [Bitbucket service endpoint](#)
- [Chef service endpoint](#)
- [Docker Host service endpoint](#)
- [Docker Registry service endpoint](#)
- [External Git service endpoint](#)
- [Generic service endpoint](#)
- [GitHub service endpoint](#)
- [Jenkins service endpoint](#)
- [Kubernetes service endpoint](#)
- [npm service endpoint](#)
- [NuGet service endpoint](#)
- [Service Fabric service endpoint](#)
- [SSH service endpoint](#)
- [Subversion service endpoint](#)
- [Team Foundation Server / VSTS service endpoint](#)
- [Visual Studio Mobile Center \(App Center\) service endpoint](#)

After you enter the parameters when creating a service endpoint, 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 endpoint

Defines and secures a connection to a Microsoft Azure subscription using Azure credentials or an Azure management certificate.

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 endpoint in task properties. This is not the name of your Azure account or subscription.
Environment	Required. Select <b>Azure Cloud</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 endpoint.

### Azure Resource Manager service endpoint

Defines and secures a connection to a Microsoft Azure subscription using Service Principal Authentication (SPA). The dialog offers two modes:

#### Automated subscription detection.

You cannot use this version of the dialog to connect to an [Azure Government Cloud](#).

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your Azure account or subscription.
Subscription	Select an existing Azure subscription. <a href="#">More information</a> .

## Manual subscription definition

You must use this version of the dialog when connecting to an [Azure Government Cloud](#).

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your Azure account or subscription.
Environment	Required. Select <b>Azure Cloud</b> or one of the pre-defined <a href="#">Azure Government Clouds</a> where your subscription is defined.
Subscription ID	Required only if you want to use an existing service principal. The GUID-like identifier for your Azure subscription (not the subscription name). <a href="#">More information</a> .
Subscription Name	Required only if you want to use an existing service principal. The name of your Microsoft Azure subscription (account). <a href="#">More information</a> .
Service Principal ID	Required only if you want to use an existing service principal. The Azure Active Directory client application ID for the account. <a href="#">More information</a> .
Service Principal Key	Required only if you want to use an existing service principal. The Azure Active Directory client authentication key for the account. <a href="#">More information</a> .
Tenant ID	Required only if you want to use an existing service principal. The ID of the client tenant in Azure Active Directory. <a href="#">More information</a> .

See [Use portal to create an Azure Active Directory application and service principal that can access resources](#).

## Restricting access rights

By default, the service endpoint will give users read/write permissions as a **Contributor** to all the resources within the specified subscription. If you prefer to restrict the access rights of users of the service endpoint, you must use the manual approach to creating the endpoint with a service principal. You can give a service principal permissions at the subscription level, resource group level, or resource level. For details of how to restrict a service principal's access rights by using Role-Based Access Control (RBAC) roles, see [Use portal to create an Azure Active Directory application and service principal that can access resources](#).

If your subscription is defined in an [Azure Government Cloud](#), ensure your application meets the relevant compliance requirements before you configure a service endpoint.

When you start to create the endpoint, the code interrogates Azure for subscriptions that are valid for the credentials you are currently signed into VSTS or TFS with. This applies to both Microsoft accounts and School or Work accounts. It displays a list of these for you to select the one you want to use.

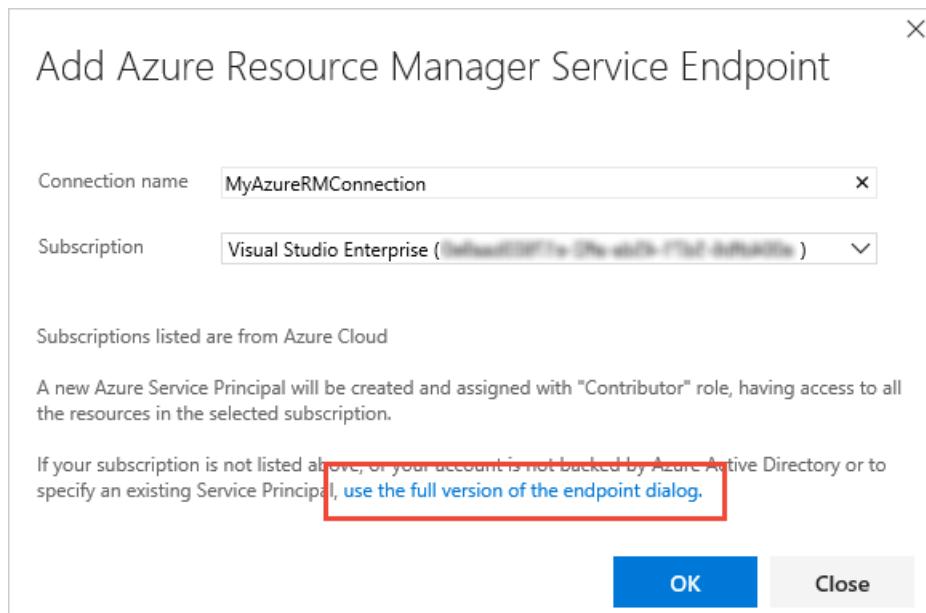
If no subscriptions are shown, or subscriptions other than the one you want to use, you must sign out of VSTS or TFS and sign in again using the appropriate account credentials. See also [Troubleshoot Azure Resource Manager service endpoints](#).

Selecting an existing subscription automatically creates a new Azure service principal that is assigned the

**Contributor** role and so has access to all resources within the subscription. You can edit this service principal in the Azure portal, **Subscriptions | Users | Roles** section. For more details, see [Azure Active Directory for developers](#).

If you want to use an existing service principal instead of creating a new one:

1. Download and run [this PowerShell script](#) in an Azure PowerShell window. When prompted, enter your subscription name, password, role (optional), and the type of cloud such as Azure Cloud (the default) or an Azure Government Cloud.
2. Switch from the simplified version of the dialog to the full version using the link in the dialog.



3. Enter a user-friendly name to use when referring to this service endpoint connection.
4. Select the Environment name (such as Azure Cloud or an Azure Government Cloud).
5. Copy these fields from the output of the PowerShell script into the Azure subscription dialog textboxes:
  - Subscription ID
  - Subscription Name
  - Service Principal ID
  - Service Principal Key
  - Tenant ID

See [this blog post](#) for details about using service principal authentication.

### Azure Service Bus service endpoint

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 endpoint in task properties. This is not the name of your Azure account or subscription.
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.

### Azure Service Fabric service endpoint

Defines and secures a connection to a Microsoft Azure Service Fabric cluster.

PARAMETER	DESCRIPTION
[authentication type]	Required. Select <b>No authentication</b> , <b>Azure Active Directory credentials</b> , or <b>Certificate based</b> .
Connection Name	Required. The name you will use to refer to this endpoint in task properties.
Cluster endpoint	Required. The client endpoint of the remote cluster to connect to. Prefix with <b>tcp://</b> .
Username	Required for Azure Active Directory authentication. The username to use when connecting to the remote cluster.
Password	Required for Azure Active Directory authentication. The password for the specified username.
Client certificate	Required for certificate based authentication. The Base64-encoded contents of the client certificate.
Password	The password for the certificate when using certificate based authentication.

You can use the following PowerShell script to obtain a Base64-encoded representation of a certificate:

```
[System.Convert]::ToString([System.IO.File]::ReadAllBytes("path-to-certificate-file\certificate.pfx"))
```

### Bitbucket service endpoint

Defines a connection to a Bitbucket server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

### Chef service endpoint

Defines and secures a connection to a [Chef](#) automation server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
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 endpoint

Defines and secures a connection to a Docker host.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
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 endpoint

Defines and secures a connection to a Docker registry.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
Docker Registry	Required. The URL of the Docker registry. A default value is provided.
Docker ID	Required. The identifier of the Docker account user.
Password	Required. The password for the account user identified above.
Email	Optional. An email address to receive notifications.

### External Git service endpoint

Defines and secures a connection to a Git repository server. Note that there is a specific endpoint for [GitHub](#).

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.

PARAMETER	DESCRIPTION
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 endpoint

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 endpoint in task properties. This is not the name of your account or subscription with the service.
Server URL	Required. The URL of the service.
User name	Required. The username to connect to the service.
Password/Token Key	Required. The password or access token for the specified username.

### GitHub service endpoint

Defines a connection to a GitHub repository. Note that there is a specific endpoint for [other Git servers](#).

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 endpoint in task properties. This is not the name of your GitHub account or subscription.

#### NOTE

If you select **Grant authorization** for the **Choose authorization** option, the dialog shows an **Authorize** button that opens the GitHub login page. If you select **Personal access token** you must obtain a suitable token and paste it into the **Token** textbox. The dialog shows the recommended scopes for the token: **repo, user, admin:repo\_hook**. See [this page](#) on GitHub for information about obtaining an access token. Then register your GitHub account in your profile:

- Open your profile from your account name at the right of the VSTS page heading.
- At the top of the left column, under **DETAILS**, choose **Security**.
- In the **Security** tab, in the right column, choose **Personal access tokens**.

- Choose the **Add** link and enter the information required to create the token.

Also see [Artifact sources](#).

### Jenkins service endpoint

Defines a connection to the Jenkins service.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
Server URL	Required. The URL of the service.
Accept untrusted SSL certificates	Set this option to allow Jenkins 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 [VSTS Integration with Jenkins](#) and [Artifact sources](#).

### Kubernetes service endpoint

Defines and secures a connection to a [Kubernetes](#) automation account.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
Server URL	Required. The URL of the Kubernetes automation service.
Kubeconfig	The contents of the kubectl configuration file.

### npm service endpoint

Defines and secures a connection to an npm server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
Registry URL	Required. The URL of the npm server.
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.

PARAMETER	DESCRIPTION
Personal Access Token	Required when connection type is <b>External VSTS</b> . The token to use to authenticate with the service. <a href="#">Learn more</a> .

## NuGet service endpoint

Defines and secures a connection to a NuGet server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
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 VSTS</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.

## Service Fabric service endpoint

Defines and secures a connection to a Service Fabric cluster.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
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> .

PARAMETER	DESCRIPTION
Cluster SPN	Required when connection type is <b>Others</b> and using Windows security.

### SSH service endpoint

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 endpoint in task properties.
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 endpoint

Defines and secures a connection to the Subversion repository.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
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 definition, use this parameter to specify the realm containing the credentials specified for this endpoint.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

### Team Foundation Server / VSTS service endpoint

Defines and secures a connection to another TFS or VSTS account.

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 endpoint in task properties. This is not the name of your account or subscription with the service.
Connection URL	Required. The URL of the TFS or VSTS instance.
User name	Required for Basic authentication. The username to connect to the service.
Password	Required for Basic authentication. The password for the specified username.
Personal Access Token	Required for Token Based authentication (TFS 2017 and newer and VSTS 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 VSTS and TFS](#).

### Visual Studio Mobile Center (App Center) service endpoint

Defines and secures a connection to Visual Studio App Center.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
API Token	Required. The token to use to authenticate with the service. <a href="#">Learn more</a> .

## Extensions for other endpoints

Other service endpoint types and tasks can be installed in VSTS and Team Foundation Server as extensions.

Some examples of service endpoints currently available through extensions are:

- [TFS artifacts for Release Management](#). Deploy on-premises TFS builds with VSTS Release Management through a TFS service endpoint connection and the **Team Build (external)** artifact, even when the TFS machine is not reachable directly from VSTS. For more information, see [External TFS](#) and [this blog post](#).
- [TeamCity artifacts for Release Management](#). This extension provides integration with TeamCity through a TeamCity service endpoint, enabling artifacts produced in TeamCity to be deployed by using Release Management. 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.

For information about creating your own custom extensions, see [Overview of extensions for VSTS](#).

## Endpoint security

You can control who can define new service endpoints in a library, and who can use an existing service endpoint.

**Roles** are defined for service endpoints, and **membership** in these roles governs the operations you can perform on those endpoints.

ROLE ON A LIBRARY SERVICE ENDPOINT	PURPOSE
User	Members of this role can use the endpoint when authoring build or release definitions.
Administrator	In addition to using the endpoint, members of this role can manage membership of all other roles for the service endpoint. The user that created the service endpoint is automatically added to the Administrator role for that service endpoint.

Two special groups called **Endpoint administrators** and **Endpoint creators** are added to every team project.

Members of the Endpoint administrators group can manage all endpoints. By default, project administrators are added as members of this group. This group is also added as an administrator to every endpoint created.

Members of the Endpoint creators group can create new endpoints. By default, project contributors are added as members of this group.

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Secure files

12/21/2017 • 1 min to read • [Edit Online](#)

## VSTS

Use the **Secure Files** library to store files such as signing certificates, Apple Provisioning Profiles, Android Keystore files, and SSH keys on the server without having to commit them to your source repository. Secure files are defined and managed in the **Library** tab of the **Build & Release** hub.

The contents of the secure files are encrypted and can only be used during the build or release process by referencing them from a task. The secure files are available across multiple build and release definitions in the team project based on the security settings. Secure files follow the [library security model](#).

There's a size limit of 10 MB for each secure file.

## Q&A

### How can I 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](#).

# Build and release retention policies

12/19/2017 • 8 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)**

Retention policies are used to configure how long builds and releases are to be retained by the system. The primary reasons to delete older builds and releases are to conserve storage and to reduce clutter. The main reasons to keep builds and releases are for audit and tracking.

## Build retention

In most cases you don't need to retain completed builds longer than a certain number of days. Using build retention policies, you can control **how many days** you want to keep each build before deleting it and the **minimum number of builds** that should be retained for each definition.

As an author of a build definition, you can customize retention policies for builds of your definition on the **Retention** tab. You can also customize these policies on a branch-by-branch basis if you are building from [Git repositories](#).

### Global build retention policy

If you are using an on-premises Team Foundation Server, you can specify build retention policy defaults and maximums for a team project collection. You can also specify when builds are permanently destroyed (removed from the **Deleted** tab in the build explorer).

If you are using VSTS, you can view but not change these settings for your account.

Global build retention policy settings can be managed from the **Build and Release** settings of your account or team project collection:

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_buildQueue](https://{your_account}.visualstudio.com/_admin/_buildQueue)
- 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 builds can be retained for all build definitions. Authors of build definitions cannot configure settings for their definitions beyond the values specified here.

The **default retention policy** sets the default retention values for all the build definitions. Authors of build definitions can override these values.

The **build destruction policy** helps you keep the builds for a certain period of time after they are deleted. This policy cannot be overridden in individual build definitions.

### Git repositories

If your [repository type](#) is one of the following, you can define multiple retention policies with branch filters:

- Git in Visual Studio Team Services (VSTS) or Team Foundation Server (TFS)
- GitHub
- External Git

For example, your team may want to keep:

- User branch builds for five days, with a minimum of a single successful or partially successful build for each branch.
- Master and feature branch builds for 10 days, with a minimum of three successful or partially successful builds for each of these branches. You exclude a special feature branch that you want to keep for a longer period of time.
- Builds from the special feature branch and all other branches for 15 days, with a minimum of a single successful or partially successful build for each branch.

The following example retention policy for a build definition meets the above requirements:

Definitions / Our build | Builds

Build Options Repository Variables Triggers General Retention History

Save Queue build... Undo

**Add new rule...**

**Days to keep:** 5 **Delete build record:**  **Branch filters:**  
**Minimum to keep:** 1 **Delete source label:**  **X Include**  `users/*`  
**Delete test results:**  **Add new filter**

**Days to keep:** 10 **Delete build record:**  **Branch filters:**  
**Minimum to keep:** 3 **Delete source label:**   
**Delete test results:**  **X Include**  `master`  
**X Include**  `features/*`  
**X Exclude**  `features/special`  
**Add new filter**

**Days to keep:** 15 **Delete build record:**  **Branch filters:**  
**Minimum to keep:** 1 **Delete source label:**   
**Delete test results:**  **X Include**  `*`  
**Add new filter**

When specifying custom policies for each definition, 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:**  **Branch filters:**  
**Delete source label:**  **X Include**  `refs/pull/*`  
**Delete file share:**  **Add new filter**  
**Delete symbols:**   
**Delete test results:**

#### 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 Minimum to keep: 10	Delete build record: true Delete source label: true Delete test results: true	Branch filters: All
---	---	---------------------

## What parts of the build get deleted

When the retention policies mark a build for deletion, you can control which information related to the build is deleted:

- Build record: You can choose to delete the entire build record or keep basic information about the build even after the build is deleted.
- Source label: If you label sources as part of the build, then you can choose to delete the tag (for Git) or the label (for TFVC) created by a build.
- Automated test results: You can choose to delete the automated test results associated with the build (for example, results published by the Publish Test Results build step).

The following information is deleted when a build is deleted:

- Logs
- [Published artifacts](#)
- [Published symbols](#)

## When are builds deleted

### VSTS

Your retention policies are processed once per day. The timing of this process varies because we spread the work throughout the day for load balancing purposes. There is no option to change this process.

### TFS

Your retention policies run every day at 3:00 A.M. UTC. There is no option to change this process.

## Release retention

The release retention policies for a release definition determine how long a release and the build linked to it are retained. Using these policies, you can control **how many days** you want to keep each release after it has been last modified or deployed and the **minimum number of releases** that should be retained for each definition. The retention timer on a release is reset every time a release is modified or deployed to an environment. The minimum number of releases to retain setting takes precedence over the number of days. For example, if you specify to retain a minimum of three releases, the most recent three will be retained indefinitely - irrespective of the number of days specified. However, you can manually delete these releases when you no longer require them.

As an author of a release definition, you can customize retention policies for releases of your definition on the **Retention** tab. You can also customize these policies on an [environment-by-environment basis](#).

## Global release retention policy

If you are using an on-premises Team Foundation Server, you can specify release retention policy defaults and maximums for a team project. You can also specify when releases are permanently destroyed (removed from the **Deleted** tab in the build explorer).

If you are using VSTS, you can view but not change these settings for your team project.

Global release retention policy settings can be managed from the **Release** settings of your team project:

- VSTS:

```
https://{{your_account}}.visualstudio.com/{{team_project}}/_admin/_apps/hub/ms.vss-releaseManagement-web.release-project-admin-hub
```

- On-premises:

```
https://{{your_server}}/tfs/{{collection_name}}/{{team_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 definitions. Authors of release definitions cannot configure settings for their definitions beyond the values specified here.

The **default retention policy** sets the default retention values for all the release definitions. Authors of build definitions 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 definitions.

In TFS, release retention management is restricted to specifying the number of days, and this is available only in TFS 2015.3 and newer.

### Environment-specific retention

You may want to retain more releases that have been deployed to specific environments. For example, your team may want to keep:

- Releases deployed to Production environment for 60 days, with a minimum of three last deployed releases.
- Releases deployed to Pre-production environment for 15 days, with a minimum of one last deployed release.
- Releases deployed to QA environment for 30 days, with a minimum of two last deployed releases.
- Releases deployed to Dev environment for 10 days, with a minimum of one last deployed release.

The following example retention policy for a release definition meets the above requirements:

The screenshot shows the 'Retention' tab of a release definition for 'Fabrikam'. The 'Retention' tab is highlighted with a red box. The left sidebar lists environments: '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 main area shows 'Settings for Dev' with fields: 'Days to retain a release \*' set to 30, 'Minimum releases to keep \*' set to 3, and a checked checkbox for 'Retain associated artifacts'. A link at the bottom says '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 definition, you cannot exceed the maximum limits set by administrator.

## Interaction between build and release retention

The build linked to a release has its own retention policy, which may be shorter than that of the release. If you want to retain the build for the same period as the release, set the **Retain build** checkbox for the appropriate environments. 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 definition, delete a release, or when the retention policy deletes a release automatically, the retention policy for the associated build will determine when that build is deleted.

In TFS, interaction between build and release retention is available in TFS 2017 and newer.

## Q&A

### Are manual test results deleted?

No

### If I mark a build or a release to be retained indefinitely, does the retention policy still apply?

No. Neither the definition's retention policy nor the maximum limits set by the administrator are applied when you mark an individual build or release to be retained indefinitely. It will remain until you stop retaining it indefinitely.

### How do I specify that builds deployed to production will be retained longer?

Customize the retention policy on the release definition. Specify the number of days that releases deployed to production must be retained. In addition, indicate that builds associated with that release are to be retained. This will override the build retention policy.

### I did not mark builds to be retained indefinitely. However, I see a large number of builds being retained. How can I prevent this?

Builds that are deployed as part of releases are also governed by the release retention policy. Customize the release retention policy as explained above.

### Are automated test results that are published as part of a release retained until the release is deleted?

Test results published within an environment of a release are associated with both the release and the build. These test results are retained as specified by the retention policy configured for the build and for the test results. If you are not deploying Team Foundation Build through Release Management, and are still publishing test results, the retention of these results is governed by the policy on test results, and is not linked to retention of releases.

# Deploy an agent on Windows

1/25/2018 • 7 min to read • [Edit Online](#)

[VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions \(XAML builds\)](#)

To build and deploy Windows, Azure, and other Visual Studio solutions you'll need at least one Windows agent. Windows agents can also build Java and Android apps.

Before you begin:

- If your code is in [VSTS](#) and a [hosted agent](#) meets your needs, you can skip setting up a private Windows agent.
- If your code is in an on-premises Team Foundation Server (TFS) 2015 server, see [Deploy an agent on Windows for on-premises TFS 2015](#).
- Otherwise, you've come to the right place to set up an agent on Windows. Continue to the next section.

## Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

## Check prerequisites

Make sure your machine is prepared with our [Windows system prerequisites](#).

If you're building from a Subversion repo, you must install the Subversion client on the machine.

## Prepare permissions

### Decide which user you'll use

Decide which user account you're going to use to register the agent.

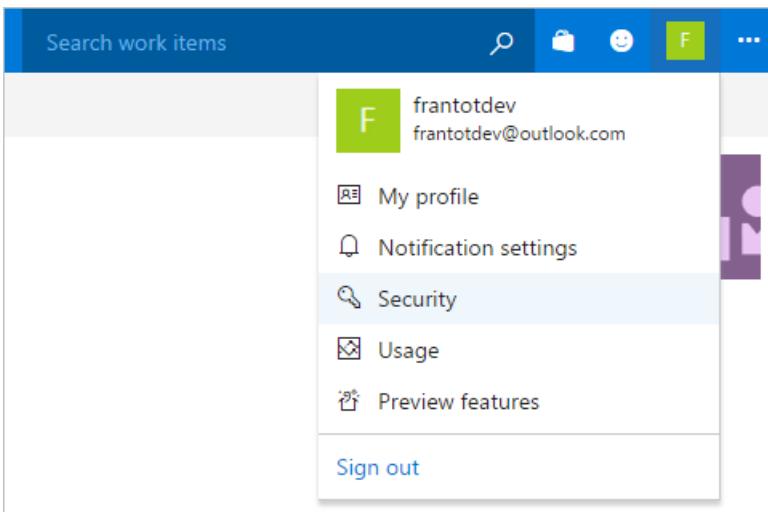
### Authenticate with a personal access token (PAT) to VSTS or TFS 2017

1. Sign in with the user account you plan to use in either your VSTS account (

`https://{your-account}.visualstudio.com` ) or your Team Foundation Server web portal (

`https://{your-server}:8080/tfs/` ).

2. From your home page, open your profile. Go to your security details.



3. Create a personal access token.

**DETAILS**

**Security**

- [Personal access tokens](#)
- [Alternate authentication credentials](#)
- [OAuth authorizations](#)
- [SSH public keys](#)

**Personal access tokens**

Personal access tokens can be used instead of a password to allo

**Add** Revoke All

You have not created any personal access tokens.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a **deployment group** agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.
5. Copy the token. You'll use this token when you configure the agent.

#### Authenticate as TFS user

- **TFS 2017 and newer:** You can use either a domain user or a local Windows user on each of your TFS application tiers.
- **TFS 2015 (applies only to macOS and Linux):** 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 you plan to use is a VSTS account owner or a TFS server administrator? If so, then skip these steps. Otherwise you might see a message like this: *Sorry, we couldn't add the identity. Please try a different identity.*

1. Open a browser and navigate to the *Agent pools* tab for your VSTS account or TFS server:

- VSTS: [https://{{your\\_account}}.visualstudio.com/\\_admin/\\_AgentPool](https://{{your_account}}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{{your\\_server}}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{{your_server}}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{{your\\_server}}:8080/tfs/\\_admin/\\_AgentPool](http://{{your_server}}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it. The

administrator can be an agent pool administrator, a [VSTS account owner](#), or a [TFS server administrator](#). If it's a [deployment group](#) agent, the administrator can be a deployment group administrator, a [VSTS account owner](#), or a [TFS server administrator](#). You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page of the **Build & Release** hub.

**Q:** I'm concerned about security. How is this account used? **A:** [Agent communication](#).

## Download and configure the agent

1. Log on to the machine using the account for which you've prepared permissions as explained above.
2. In your web browser, sign on to VSTS or TFS, and navigate to the **Agent pools** tab:

- VSTS: `https://{{your_account}}.visualstudio.com/_admin/_AgentPool`
- TFS 2017 and newer: `https://{{your_server}}/tfss/DefaultCollection/_admin/_AgentPool`
- TFS 2015: `http://{{your_server}}:8080/tfs/_admin/_AgentPool`

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

3. Click **Download agent**.
4. On the **Get agent** dialog box, click **Windows**.
5. Click the **Download** button.
6. Follow the instructions on the page.

### Server URL

- VSTS: `https://{{your-account}}.visualstudio.com`
- TFS 2017 and newer: `https://{{your_server}}/tfss`
- TFS 2015: `http://{{your-server}}:8080/tfs`

### Authentication type

#### VSTS

Choose **PAT**, and then paste the [PAT token you created](#) into the command prompt window.

#### TFS

##### IMPORTANT

Make sure your server is [configured to support the authentication method](#) you want to use.

When you configure your agent to connect to TFS, you've got the following options:

- **Alternate** Connect to TFS using Basic authentication. After you select Alternate you'll be prompted for your credentials.
- **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 VSTS and TFS 2017 and newer. After you choose PAT, 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 VSTS 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 configured the agent to run interactively, to run it:

```
.\run.cmd
```

If you configured the agent to run as a service, it starts automatically. You can view and control the agent running status from the services snap-in. Run `services.msc` and look for "VSTS Agent (*name of your agent*)".

If you need to change the logon account, don't do it from the services snap-in. Instead, see the information below to re-configure the agent.

## Replace an agent

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](#).

## 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 dev machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

## **IMPORTANT**

After you install new software on a agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

## **Q&A**

### **I'm running a firewall and my code is in VSTS. 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:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{accountname}}.visualstudio.com  
https://{{accountname}}.vsrm.visualstudio.com  
https://{{accountname}}.pkgs.visualstudio.com  
https://{{accountname}}.vssps.visualstudio.com
```

### **How do I run the agent with self-signed certificate?**

[Run the agent with self-signed certificate](#)

### **How do I run the agent behind a web proxy?**

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

### **How do I configure the agent to bypass a web proxy and connect to VSTS?**

If you want the agent to bypass your proxy and connect to VSTS directly, then you should configure your web proxy to enable the agent to access the following URLs:

- `https://management.core.windows.net`
- `*.visualstudio.com`
- `https://login.microsoftonline.com`
- `https://app.vsspsext.visualstudio.com`

## **NOTE**

This procedure enables the agent to bypass a web proxy. Your build definition 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 [VSTS](#) 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

12/19/2017 • 6 min to read • [Edit Online](#)

[VSTS](#) | [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 [VSTS](#) or TFS 2017 and newer, then you need to use a newer agent. See [Deploy an agent on Windows](#).
- If you use TFS, you might already have a build and release agent running. An agent is automatically or optionally deployed in some cases when you [set up Team Foundation Server](#).
- Otherwise, you've come to the right place to set up an agent on Windows for TFS 2015. Continue to the next section.

## Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

## Check prerequisites

Before you begin, make sure your agent machine is prepared with these prerequisites:

- An [operating system that is supported by Visual Studio 2013](#) or newer
- Visual Studio 2013 or Visual Studio 2015
- PowerShell 3 or newer ([Where can I get a newer version of PowerShell?](#))

## Download and configure the agent

1. Make sure you're logged on the machine as an agent pool administrator. See [Agent pools](#).
2. Navigate to the **Agent pools** tab: `http://{your_server}:8080/tfs/_admin/_AgentPool`
3. Click **Download agent**.
4. Unzip the .zip file into the folder on disk from which you would like to run the agent. To avoid "path too long" issues on the file system, keep the path short. For example: `C:\Agent\`
5. Run Command Prompt as Administrator, and then run:

```
ConfigureAgent.cmd
```

6. Respond to the prompts.

### Choose interactive or service mode

For guidance on whether to run the agent in interactive mode or as a service, see [Agents: Interactive vs. service](#).

[Run as a service](#)

If you chose to run the agent as a Windows service, then the agent running status can be controlled from the Services snap-in. Run services.msc and look for "VSO Agent (<name of your agent>)".

If you need to change the logon account, don't do it from the services snap-in. Instead, from an elevated Command Prompt, run:

```
C:\Agent\Agent\VsoAgent.exe /ChangeWindowsServiceAccount
```

#### Run interactively

If you chose to run interactively, then to run the agent:

```
C:\Agent\Agent\VsoAgent.exe
```

## Command-line parameters

You can use command-line parameters when you configure the agent (`ConfigureAgent.cmd`) and when you run the agent (`Agent\VsoAgent.exe`). These are useful to avoid being prompted during unattended installation scripts and for power users.

### Common parameters

`/Login:UserName,Password[;AuthType=(AAD|Basic|PAT)]`

Used for configuration commands against a VSTS account. 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://example.visualstudio.com`

#### */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 dev machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

#### **IMPORTANT**

After you install new software on a agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

## Q&A

### What version of PowerShell do I need? Where can I get a newer version?

The Windows Agent requires PowerShell version 3 or later. To check your PowerShell version:

```
$PSVersionTable.PSVersion
```

If you need a newer version of PowerShell, you can download it:

- PowerShell 3: Windows Management Framework 3.0
- PowerShell 4: Windows Management Framework 4.0
- PowerShell 5: Windows Management Framework 5.0

### Why do I get this message when I try to queue my build?

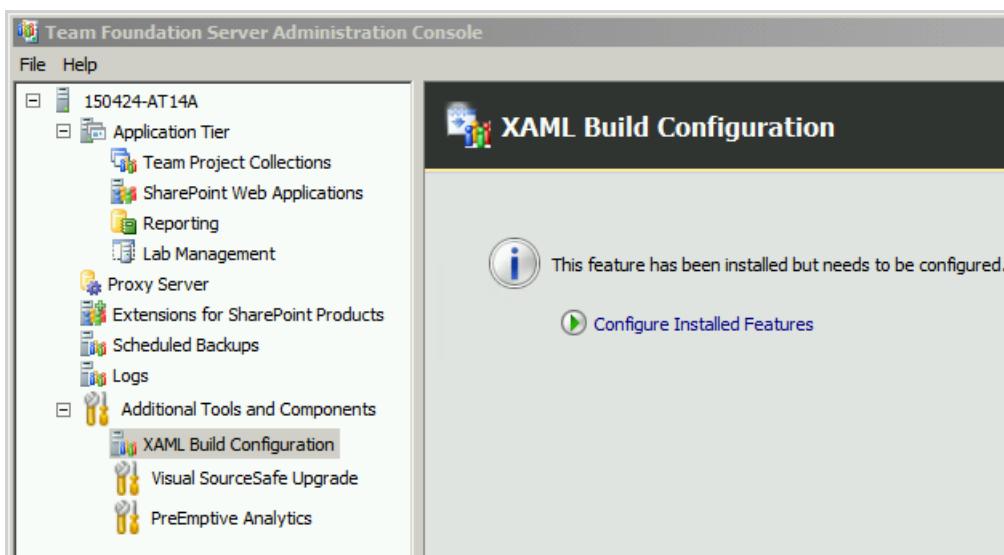
When you try to queue a build or a deployment, you might get a message such as: "No agent could be found with the following capabilities: msbuild, vsvisualstudio, vstest." One common cause of this problem is that you need to install prerequisite software such as Visual Studio on the machine where the build agent is running.

### What version of the agent runs with my version of TFS?

TFS VERSION	AGENT VERSION
2015 RTM	1.83.2
2015.1	1.89.0
2015.2	1.95.1
2015.3	1.95.3

### Can I still configure and use XAML build controllers and agents?

Yes. If you are an existing customer with custom build processes you are not yet ready to migrate, you can continue to use XAML builds, controllers, and agents.



### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Deploy an agent on macOS

1/25/2018 • 10 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

To build and deploy Xcode apps or Xamarin.iOS projects, you'll need at least one macOS agent. This agent can also build and deploy Java and Android apps.

Before you begin:

- If your build and release definitions are in [VSTS](#) and a [hosted agent](#) meets your needs, you can skip setting up a private macOS agent.
- Otherwise, you've come to the right place to set up an agent on macOS. Continue to the next section.

## Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

## Check prerequisites

Make sure your machine is prepared with our [macOS system prerequisites](#).

## Prepare permissions

If you're building from a Subversion repo, you must install the Subversion client on the machine.

### Decide which user you'll use

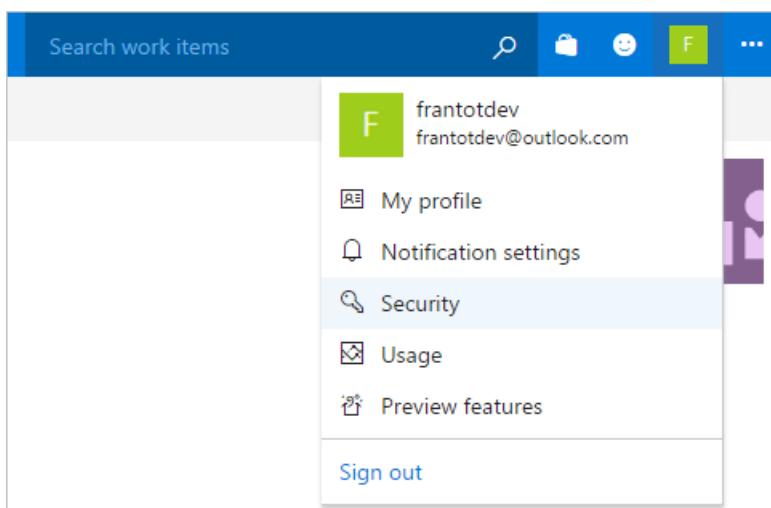
Decide which user account you're going to use to register the agent.

### Authenticate with a personal access token (PAT) to VSTS or TFS 2017

1. Sign in with the user account you plan to use in either your VSTS account (

<https://{{your-account}}.visualstudio.com>) or your Team Foundation Server web portal (  
<https://{{your-server}}:8080/tfs/>).

2. From your home page, open your profile. Go to your security details.



3. Create a personal access token.

**Personal access tokens**

Personal access tokens can be used instead of a password to allo

Add      Revoke All

You have not created any personal access tokens.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a **deployment group** agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.
5. Copy the token. You'll use this token when you configure the agent.

#### Authenticate as TFS user

- **TFS 2017 and newer:** You can use either a domain user or a local Windows user on each of your TFS application tiers.
- **TFS 2015 (applies only to macOS and Linux):** 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 you plan to use is a VSTS account owner or a TFS server administrator? If so, then skip these steps. Otherwise you might see a message like this: *Sorry, we couldn't add the identity. Please try a different identity.*

1. Open a browser and navigate to the *Agent pools* tab for your VSTS account or TFS server:
  - VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
  - TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
  - TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)
2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator can be an agent pool administrator, a **VSTS account owner**, or a **TFS server administrator**. If it's a **deployment group** agent, the administrator can be an deployment group administrator, a **VSTS account owner**, or a **TFS server administrator**. You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page of the **Build & Release** hub.

**Q:** I'm concerned about security. How is this account used? **A:** [Agent communication](#).

## Download and configure the agent

#### VSTS and TFS 2017 and newer

1. Log on to the machine using the account for which you've prepared permissions as explained above.
2. In your web browser, sign on to VSTS or TFS, and navigate to the **Agent pools** tab:
  - VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
  - TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
  - TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

The TFS URL doesn't work for me. How can I get the correct URL?

3. Click **Download agent**.
4. On the **Get agent** dialog box, click **OS X**.
5. Click the **Download** button.
6. Follow the instructions on the page.

## 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

- VSTS: <https://{your-account}.visualstudio.com>
- TFS 2017 and newer: [https://{your\\_server}/tfss](https://{your_server}/tfss)
- TFS 2015: [http://{your\\_server}:8080/tfs](http://{your_server}:8080/tfs)

## Authentication type

### VSTS

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 VSTS or TFS](#).

### TFS

#### IMPORTANT

Make sure your server is [configured to support the authentication method](#) you want to use.

When you configure your agent to connect to TFS, you've got the following options:

- **Alternate** Connect to TFS using Basic authentication. After you select Alternate you'll be prompted for your credentials.
- **Integrated** Not supported on macOS or Linux.
- **Negotiate** (Default) Connect to TFS as a user other than the signed-in user via a Windows authentication scheme such as NTLM or Kerberos. After you select Negotiate you'll be prompted for credentials.
- **PAT** Supported only on VSTS and TFS 2017 and newer. After you choose PAT, 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 on newer versions of TFS. Learn more at [Communication with VSTS 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
```

# Run as a launchd service

We provide the `./svc.sh` script for you to run and manage your agent as a launchd LaunchAgent service. 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:

- VSTS: the name of your account. For example if you connect to `https://fabrikam.visualstudio.com`, 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.{tfss-name}.{agent-name}
status vsts.agent.{tfss-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.agent.{tfss-name}.{agent-name}.plist

Started:
13472 0 vsts.agent.{tfss-name}.{agent-name}
```

The left number is the pid if the service is running. If second number is not zero, then a problem occurred.

### Status

Command:

```
./svc.sh status
```

Output:

```
status vsts.agent.{tfss-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.{tfss-name}.{agent-name}.testsrv.plist

Started:
13472 0 vsts.agent.{tfss-name}.{agent-name}
```

The left number is the pid if the service is running. If second number is not zero, then a problem occurred.

### Stop

Command:

```
./svc.sh stop
```

Output:

```
stopping vsts.agent.{tfss-name}.{agent-name}
status vsts.agent.{tfss-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.{tfss-name}.{agent-name}.testsrv.plist

Stopped
```

### Uninstall

You should stop before you uninstall.

Command:

```
./svc.sh uninstall
```

### Automatic log on and lock

The service runs when the user logs in. If you want the agent service start when the machine restarts, you can

configure the machine it to automatically log on and lock on startup. See [Auto Logon and Lock](#).

## Update environment variables

When you start the service, it takes a snapshot of your PATH other useful environment variables such as PATH, LANG, JAVA\_HOME, ANT\_HOME, and MYSQL\_PATH. If you need to update the variables (for example, after installing some new software):

1.

```
./env.sh
```

2.

```
./svc.sh stop
```

3.

```
./svc.sh start
```

## 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](#)

## Replace an agent

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](#).

## 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 dev machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

### IMPORTANT

After you install new software on a agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

## Q&A

### Where can I learn more about how the launchd service works?

[Apple Developer Library: Creating Launch Daemons and Agents](#)

[launchd.plist manpage](#)

### I'm running a firewall and my code is in VSTS. 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:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{accountname}}.visualstudio.com  
https://{{accountname}}.vsrm.visualstudio.com  
https://{{accountname}}.pkgs.visualstudio.com  
https://{{accountname}}.vssps.visualstudio.com
```

### How do I run the agent with self-signed certificate?

[Run the agent with self-signed certificate](#)

### How do I run the agent behind a web proxy?

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

### How do I configure the agent to bypass a web proxy and connect to VSTS?

If you want the agent to bypass your proxy and connect to VSTS directly, then you should configure your web proxy to enable the agent to access the following URLs:

- `https://management.core.windows.net`
- `*.visualstudio.com`
- `https://login.microsoftonline.com`
- `https://app.vsspsext.visualstudio.com`

#### NOTE

This procedure enables the agent to bypass a web proxy. Your build definition 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 [VSTS](#) 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 Linux

1/25/2018 • 9 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

To build or deploy you'll need at least one agent. A Linux agent can build and deploy different kinds of apps, including Java and Android apps. We support Ubuntu, Red Hat, and CentOS.

Before you begin:

- If your build and release definitions are in [VSTS](#) and a [hosted agent](#) meets your needs, you can skip setting up a private Linux agent.
- Otherwise, you've come to the right place to set up an agent on Linux. Continue to the next section.

## Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see [Build and release agents](#).

## Check prerequisites

Where are your builds and releases running?

- **VSTS:** The agent is based on CoreCLR 2.0. You can run this agent on several Linux distributions. Make sure your machine is prepared with [our prerequisites](#).
- **TFS 2018 RTM and older:** The agent is based on CoreCLR 1.0. Make sure your machine is prepared with our prerequisites for either of the supported distributions:
  - [Ubuntu systems](#)
  - [Red Hat/CentOS systems](#)

### Subversion

If you're building from a Subversion repo, you must install the Subversion client on the machine.

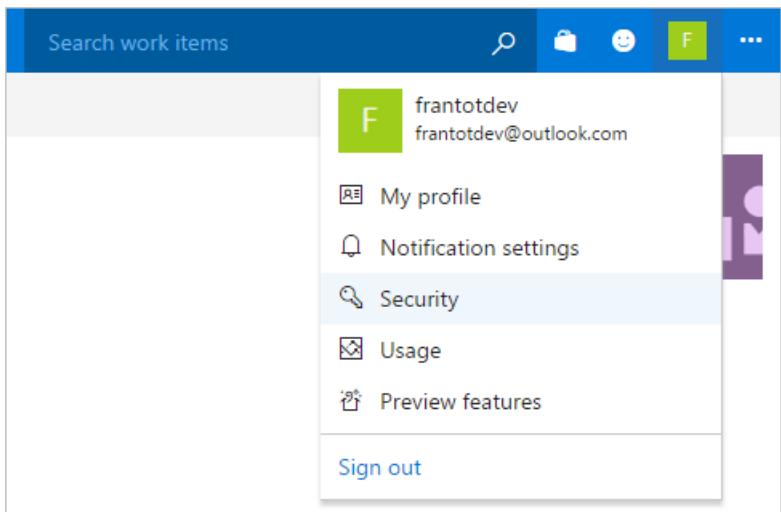
## Prepare permissions

### Decide which user you'll use

Decide which user account you're going to use to register the agent.

#### Authenticate with a personal access token (PAT) to VSTS or TFS 2017

1. Sign in with the user account you plan to use in either your VSTS account (  
<https://{{your-account}}.visualstudio.com>) or your Team Foundation Server web portal (  
<https://{{your-server}}:8080/tfs/>).
2. From your home page, open your profile. Go to your security details.



3. Create a personal access token.

**DETAILS**

**Security**

- Personal access tokens
- Alternate authentication credentials
- OAuth authorizations
- SSH public keys

**Personal access tokens**

Personal access tokens can be used instead of a password to allo

**Add** Revoke All

You have not created any personal access tokens.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a **deployment group** agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

5. Copy the token. You'll use this token when you configure the agent.

#### Authenticate as TFS user

- **TFS 2017 and newer:** You can use either a domain user or a local Windows user on each of your TFS application tiers.
- **TFS 2015 (applies only to macOS and Linux):** 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 you plan to use is a VSTS account owner or a TFS server administrator? If so, then skip these steps. Otherwise you might see a message like this: *Sorry, we couldn't add the identity. Please try a different identity.*

1. Open a browser and navigate to the *Agent pools* tab for your VSTS account or TFS server:

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator can be an agent pool administrator, a **VSTS account owner**, or a **TFS server administrator**. If it's a

deployment group agent, the administrator can be an deployment group administrator, a [VSTS account owner](#), or a [TFS server administrator](#). You can add a user to the deployment group adminstrator role in the **Security** tab on the **Deployment Groups** page of the **Build & Release** hub.

**Q:** I'm concerned about security. How is this account used? **A:** [Agent communication](#).

## Download and configure the agent

### VSTS and TFS 2017 and newer

1. Log on to the machine using the account for which you've prepared permissions as explained above.
2. In your web browser, sign on to VSTS or TFS, and navigate to the **Agent pools** tab:

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

3. Click **Download agent**.
4. On the **Get agent** dialog box, click **Linux**.
5. Click the **Download** button.
6. Follow the instructions on the page.

### 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

- VSTS: <https://{your-account}.visualstudio.com>
- TFS 2017 and newer: [https://{your\\_server}/tfss](https://{your_server}/tfss)
- TFS 2015: [http://{your\\_server}:8080/tfs](http://{your_server}:8080/tfs)

### Authentication type

#### VSTS

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 VSTS or TFS](#).

#### TFS

#### IMPORTANT

Make sure your server is [configured to support the authentication method](#) you want to use.

When you configure your agent to connect to TFS, you've got the following options:

- **Alternate** Connect to TFS using Basic authentication. After you select Alternate you'll be prompted for your credentials.
- **Integrated** Not supported on macOS or Linux.
- **Negotiate** (Default) Connect to TFS as a user other than the signed-in user via a Windows authentication scheme such as NTLM or Kerberos. After you select Negotiate you'll be prompted for credentials.
- **PAT** Supported only on VSTS and TFS 2017 and newer. After you choose PAT, 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 on newer versions of TFS. Learn more at [Communication with VSTS 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
```

## Run as a systemd service

If your agent is running on these operating systems you can run the agent as a systemd service:

- Ubuntu 16 LTS or newer
- Red Hat 7.1 or newer

We provide the `./svc.sh` script for you to run and manage your agent as a systemd service.

#### NOTE

If you have a different distribution, or if you prefer other approaches, you can use whatever kind of service mechanism you prefer. See [Service files](#).

## Commands

### Change to the agent directory

For example, if you installed in the `myagent` subfolder of your home directory:

```
cd ~/myagent$
```

### Install

Command:

```
sudo ./svc.sh install
```

This command creates a service file that points to `./runsvc.sh`. This script sets up the environment (more details below) and starts the agents host.

#### Start

```
sudo ./svc.sh start
```

#### Status

```
sudo ./svc.sh status
```

#### Stop

```
sudo ./svc.sh stop
```

#### Uninstall

You should stop before you uninstall.

```
sudo ./svc.sh uninstall
```

#### Update environment variables

When you start the service, it takes a snapshot of your PATH other useful environment variables such as PATH, LANG, JAVA\_HOME, ANT\_HOME, and MYSQL\_PATH. If you need to update the variables (for example, after installing some new software):

1.

```
./env.sh
```

2.

```
sudo ./svc.sh stop
```

3.

```
sudo ./svc.sh start
```

#### 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:

- VSTS: the name of your account. For example if you connect to `https://fabrikam.visualstudio.com`, then the service name would be `/etc/systemd/system/vsts.agent.fabrikam.our-linux-agent.service`
- TFS: the name of your on-premises TFS AT server. For example if you connect to `http://our-server:8080/tfs`, then the service name would be `/etc/systemd/system/vsts.agent.our-server.our-linux-agent.service`

```
sudo ./svc.sh install
```

 generates this file from this template: `./bin/vsts.agent.service.template`

## .service file

`sudo ./svc.sh start` finds the service by reading the `.service` file, which contains the name of systemd service file described above.

## Alternative service mechanisms

We provide the `./svc.sh` script as a convenient way for you to run and manage your agent as a systemd service. But you can use whatever kind of service mechanism you prefer (for example: initd or upstart).

You can use the template described above as to facilitate generating other kinds of service files.

## Replace an agent

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](#).

## 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 dev machine.

For example, if your build includes the [npm task](#), then the build won't run unless there's a build agent in the pool that has npm installed.

**IMPORTANT**

After you install new software on a agent, you must restart the agent for the new capability to show up in the pool so that the build can run.

## Q&A

### 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 VSTS. 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:

```
https://login.microsoftonline.com  
https://app.vssps.visualstudio.com  
https://{{accountname}}.visualstudio.com  
https://{{accountname}}.vsrm.visualstudio.com  
https://{{accountname}}.pkgs.visualstudio.com  
https://{{accountname}}.vssps.visualstudio.com
```

### How do I run the agent with self-signed certificate?

[Run the agent with self-signed certificate](#)

### How do I run the agent behind a web proxy?

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

### How do I configure the agent to bypass a web proxy and connect to VSTS?

If you want the agent to bypass your proxy and connect to VSTS directly, then you should configure your web proxy to enable the agent to access the following URLs:

- `https://management.core.windows.net`
- `*.visualstudio.com`
- `https://login.microsoftonline.com`
- `https://app.vsspsext.visualstudio.com`

**NOTE**

This procedure enables the agent to bypass a web proxy. Your build definition 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 [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Run a private agent behind a web proxy

12/5/2017 • 3 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

This topic explains how to run a v2 private agent behind a web proxy.

## VSTS, TFS 2018 RTM and newer

(Applies to agent version 2.122 and newer.)

When your private agent is behind a web proxy, you can:

- Allow your agent to connect to VSTS or TFS behind a web proxy.
- Allow your agent to get sources in the build job and download artifacts in the release job.
- Develop custom vsts-task-lib tasks that leverage the proxy agent configuration.

To enable the agent to run behind a web proxy

1. Pass `--proxyurl`, `--proxyusername` and `--proxypassword` during agent configuration.

For example:

- [Windows](#)
- [macOS and Linux](#)

```
./config.cmd --proxyurl http://127.0.0.1:8888 --proxyusername "1" --proxypassword "1"
```

We store your proxy credential securely on each platform.

- [Windows](#)
- [macOS and Linux](#)

Windows Credential Store

### 2. Limitation of agent version 122.0

This applies to TFS 2018 RTM. It also could apply to VSTS, unless you upgrade your agent.

- [Windows](#)
- [macOS and Linux](#)

The Windows Credential Store will cause a limitation for configuring agent as Windows service\*\*

Since we store your proxy credential into `Windows Credential Store` and the `Windows Credential Store` is per user, when you configure the agent as Windows service, you need run the configuration as the same user as the service is going to run as.

Ex, in order to configure the agent service run as `mydomain\buildadmin`, you need either login the box as `mydomain\buildadmin` and run `config.cmd` or login the box as someone else but use `Run as different user` option when you run `config.cmd` to run as `mydomain\buildadmin`

### How the agent handles the proxy within a build or release job

After configuring proxy for agent, agent infrastructure will start talk to VSTS/TFS service through the web proxy

specified in the `.proxy` file.

Since the code for `Get Source` step in build job and `Download Artifact` step in release job are also baked into agent, those steps will also follow the agent proxy configuration from `.proxy` file.

Agent will expose proxy configuration via environment variables for every task execution, task author need to use `vsts-task-lib` methods to retrieve back proxy configuration and handle proxy with their task.

### Get proxy configuration by using [VSTS-Task-Lib](#) method

See [VSTS-Task-Lib doc](#) for details.

## TFS 2017.2 and older

(You also can use this method for VSTS and newer versions of TFS, but we recommend the above method in those cases.)

In the agent root directory, create a `.proxy` file with your proxy server url.

- [Windows](#)
- [macOS and Linux](#)

```
echo http://name-of-your-proxy-server:8888 | Out-File .proxy
```

If your proxy doesn't require authentication, then you're ready to configure and run the agent. See [Deploy an agent on Windows](#).

#### NOTE

For backwards compatibility, if the proxy is not specified as described above, the agent also checks for a proxy URL from the `VSTS_HTTP_PROXY` environment variable.

## Proxy authentication

If your proxy requires authentication, the simplest way to handle it is to grant permissions to the user under which the agent runs. Otherwise, you can provide credentials through environment variables. When you provide credentials through environment variables, the agent keeps the credentials secret by masking them in job and diagnostic logs. To grant credentials through environment variables, set the following variables:

- [Windows](#)
- [macOS and Linux](#)

```
$env:VSTS_HTTP_PROXY_USERNAME = "proxyuser"  
$env:VSTS_HTTP_PROXY_PASSWORD = "proxypassword"
```

#### NOTE

This procedure enables the agent infrastructure to operate behind a web proxy. Your build definition 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 under agent root to specify proxy bypass Url's Regex (ECMAScript syntax).

For example:

github\.com  
bitbucket\.com

# Run the agent with self-signed certificate (on-premises TFS only)

1/25/2018 • 3 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017**

This topic explains how to run a v2 private agent with self-signed certificate.

## Work with SSL server certificate (on-premises TFS only)

```
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 (on-premises TFS and Windows agent only)**

We ship git.exe as part of windows agent, we use this git.exe for all Git related operation. When you have a self-signed SSL certificate for your on-premises TFS server, make sure to configure the git.exe we shipped to allow that self-signed SSL certificate. There are 2 approaches to solve the problem.

1. Set the following git config in global level by the agent's run as user.

```
git config --global http."https://tfss.com/".sslCAInfo certificate.pem
```

**NOTE**

Setting system level git config is not reliable on Windows, since the system level .gitconfig file is stored at the copy of git.exe we packaged which will get replaced whenever the agent is upgraded to a new version.

2. Enable git to use SChannel during configuration with 2.129.0 or higher version agent. Pass `--gituseschannel` during agent configuration

```
./config.cmd --gituseschannel
```

**NOTE**

Git SChannel has more restrictive requirements for your self-signed certificate. Self-signed certificate generated by IIS or PowerShell command may not be capable with SChannel.

**Work with SSL client certificate (on-premises TFS only)**

IIS has a SSL setting that requires all incoming requests to TFS must present client certificate in addition to the regular credential.

When that IIS SSL setting is enabled, you need to use `2.125.0` or above version agent and follow these extra steps in order to configure the build machine against your TFS server.

- Prepare all required certificate informations
  - CA certificate(s) in `.pem` format (This should contain the public key and signature of the CA certificate, you need put the root ca certificate and all your intermediate ca certificates into one `.pem` file)
  - Client certificate in `.pem` format (This should contain the public key and signature of the Client certificate)
  - Client certificate private key in `.pem` format (This should contain only the private key of the Client certificate)
  - Client certificate archive package in `.pfx` format (This should contain the signature, public key and private key of the Client certificate)
  - Use `SAME` password to protect Client certificate private key and Client certificate archive package, since they both have client certificate's private key
- Install CA certificate(s) into machine certificate store

- Windows: Windows certificate store
- Linux: OpenSSL certificate store
- macOS: System or User Keychain
- 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"
```

We store your client cert private key password securely on each platform.

```
Windows: Windows Credential Store  
macOS: macOS Keychain  
Linux: Encrypted with a symmetric key based on the machine ID
```

Click [here](#) for more detail information about agent client certificate support.

# Configure TFS authentication for your private build and release agents

12/19/2017 • 1 min to read • [Edit Online](#)

[TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#) | [Previous versions \(XAML builds\)](#)

When you deploy a private agent, you choose how the agent will authenticate to your Team Foundation Server (TFS). Here we'll show you how to configure TFS to enable your agents to use different authentication methods.

## Log on to your server

Log on to the machine where you are running TFS.

## Configure authentication

### Alternate

Configure basic authentication. See <https://github.com/Microsoft/tfs-cli/blob/master/docs/configureBasicAuth.md>.

### Integrated

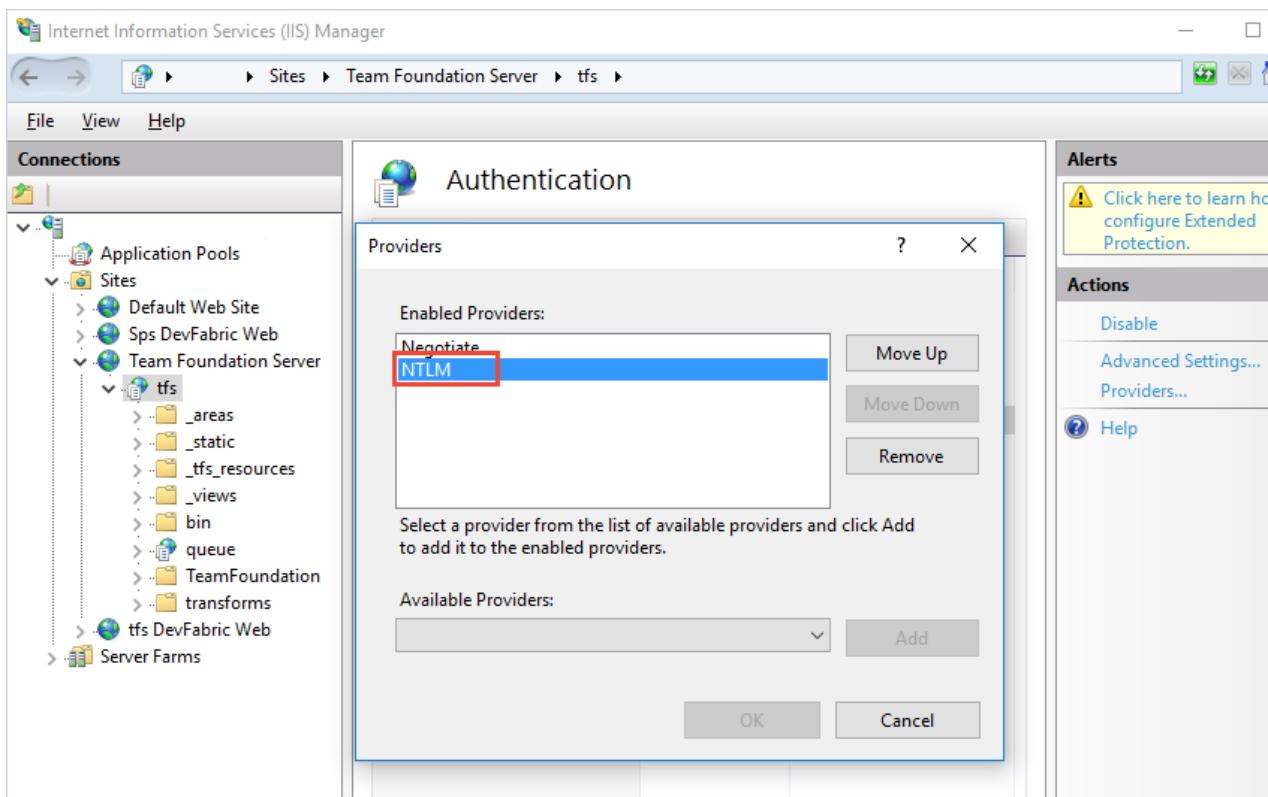
Start Internet Information Services (IIS) Manager. Select your TFS site and make sure Windows Authentication is enabled with a valid provider such as NTLM or Kerberos.

The screenshot shows the IIS Manager interface with the following details:

- Connections:** Shows the tree structure: Application Pools, Sites, Team Foundation Server, and the tfs site.
- Authentication:** The main pane displays the "Authentication" settings for the tfs site. The table shows the following configuration:

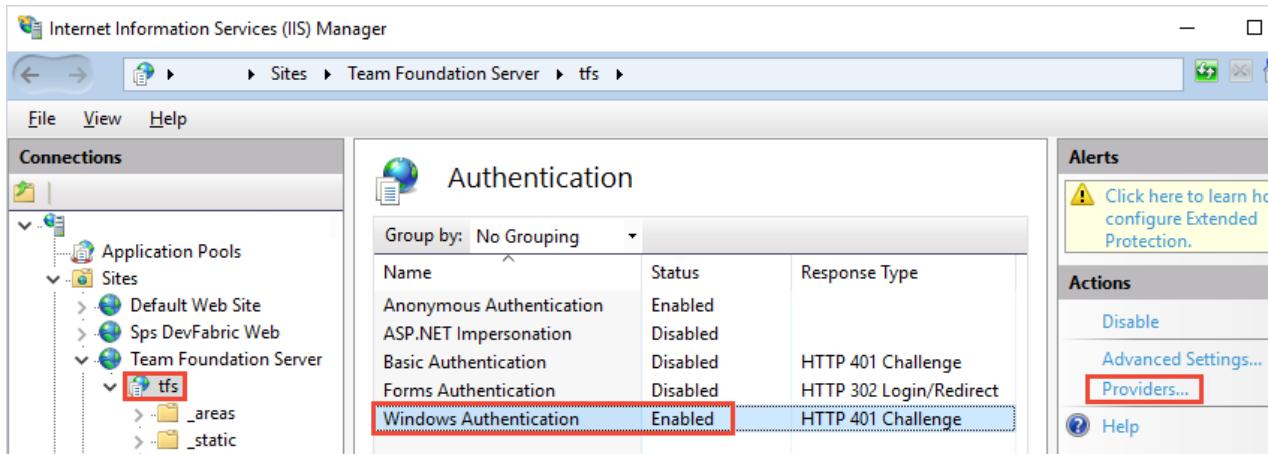
Name	Status	Response Type
Anonymous Authentication	Enabled	HTTP 401 Challenge
ASP.NET Impersonation	Disabled	HTTP 302 Login/Redirect
Basic Authentication	Disabled	HTTP 401 Challenge
Forms Authentication	Disabled	HTTP 302 Login/Redirect
Windows Authentication	Enabled	HTTP 401 Challenge

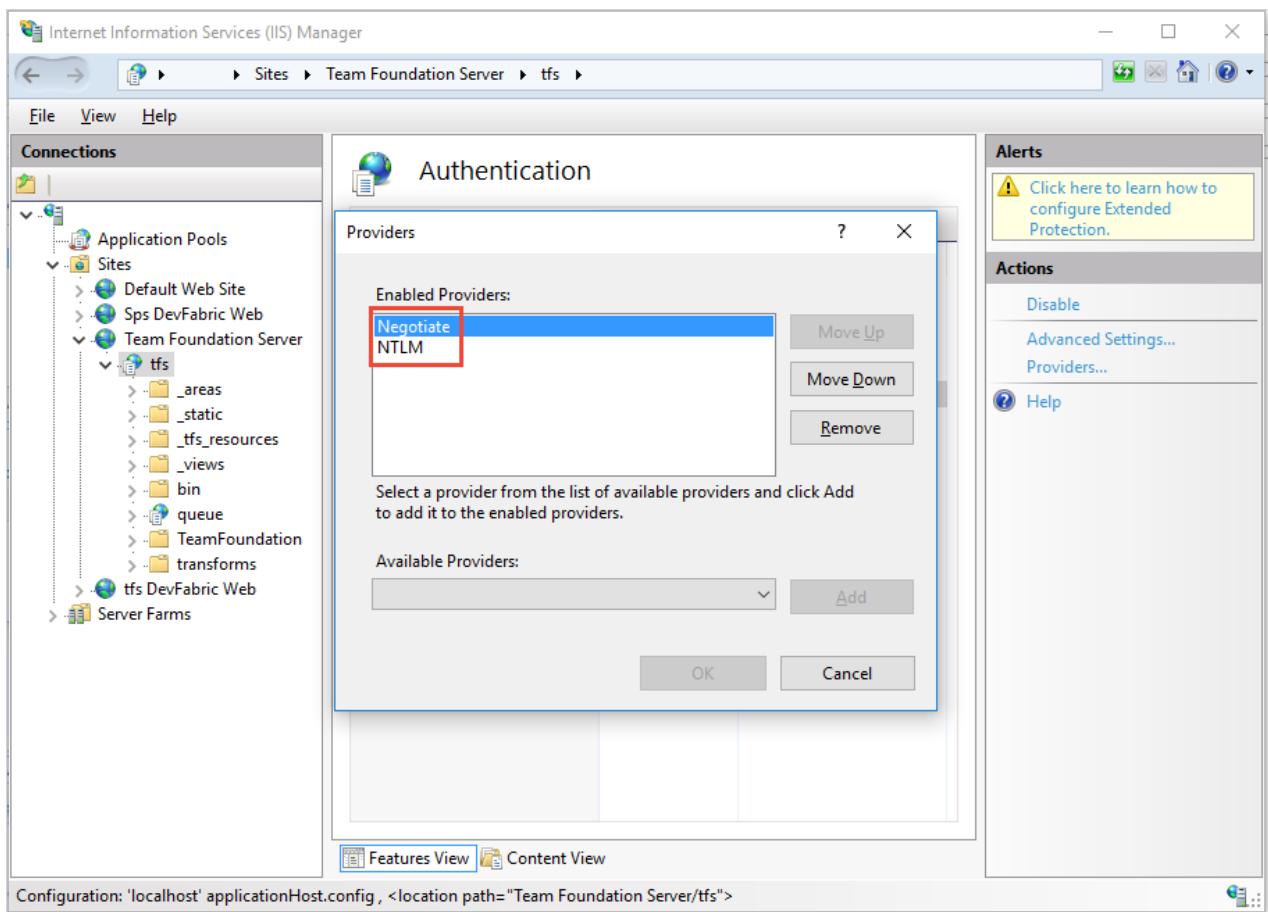
- Actions:** On the right, there are buttons for "Disable", "Advanced Settings...", and "Providers...". The "Providers..." button is highlighted with a red box.



## Negotiate

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.





## PAT

Personal access token (PAT) authentication is available in TFS 2015.3 and newer. To use a PAT, your server must be configured with HTTPS. See [Web site settings and security](#).

## Deploy your agent

### TFS 2017 and newer

- [Windows agent v2](#)
- [macOS agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [RedHat agent](#)

### TFS 2015

- [Windows agent v1](#)
- [macOS agent](#)
- [Ubuntu 14.04 agent](#)
- [Ubuntu 16.04 agent](#)
- [RedHat agent](#)

# How to use YAML builds

2/23/2018 • 3 min to read • [Edit Online](#)

## VSTS

When you define a CI build on VSTS, you've got a fundamental choice: use a web-based interface or configure your CI process as code in a YAML build. YAML build definitions give you the advantages of configuration as code.

### NOTE

To use this capability, you must have the **Build YAML definitions** [preview feature](#) enabled on your account.

In a YAML build definition, your CI build process configured as code, which means:

- The definition is versioned with your code and follows the same branching structure as your code. So you get validation of your changes through code reviews in pull requests and branch build policies.
- If a change to the build process causes a break or results in an unexpected outcome, you can much more easily identify the issue because the change is in version control with the rest of your codebase. This way you can more clearly see the issue and fix it like any other kind of bug.

## Get started

If you're new to YAML builds, or to VSTS, we suggest you begin learning with either our [ASP.NET Core](#) or [Node.js](#) tutorial.

## How do YAML builds compare to web-interface builds?

CAPABILITY	YAML	WEB INTERFACE
<b>Repository</b>		
Git and GitHub	Yes	Yes
Bitbucket	Not yet	Yes
TFVC	No	Yes
External Git, Subversion	No	Yes
<b>Capabilities</b>		
Configuration as code	Yes	No
Fan out and fan in	Yes	Not yet
Variables across phases	Yes	Not yet
Test and debug the process locally	Not yet	No

CAPABILITY	YAML	WEB INTERFACE
Set environment variables for scripts (useful for secrets)	Yes	Not yet
Badge enabled	Not yet	Yes
Pause/disable definition from web editor	Not yet	Yes
Link work items	Not yet	Yes
Create work item on failure	Not yet	Yes
Allow scripts to access OAuth token	Not yet	Yes
Build job authorization scope	Not yet	Yes
Checkout options: Tag sources, Report build status, Checkout submodules	Not yet	Yes

## Automatically create a YAML build definition

To make it more convenient to create YAML build definitions, VSTS automatically creates a definition when you add a file named `.vsts-ci.yml` to the root of your repository. It creates the build definition in a folder that has the same name as your repository.

1. Navigate to the **Code** hub, choose the **Files** tab, and then choose the repository you created in the above steps.
2. In the root folder of the repo, create a new file called **.vsts-ci.yml**.
3. Paste the following content into the file:

```
steps:
- script: echo hello world
```

If your code is in VSTS, then a new build is automatically created and queued.

### NOTE

If your team project already has a build definition that's pointing to the file, then the system does not automatically create another build definition.

## Manually create a YAML build definition

If your code is in GitHub, or if you want to create multiple YAML build definitions, then after you have published the YAML file in your repo, you can manually create a build definition.

1. Navigate to the **Builds** tab of the **Build and Release** hub in VSTS or TFS, and then click **+ New**. You are asked to **Select a template** for the new build definition.
2. Choose **YAML**, and then click **Apply**.

3. Click **Get sources**.
4. Select the repo that contains your .YML file.
5. Click **Process**.
6. For the **Agent queue** select any of the hosted options.
7. For YAML path, click the ... button, and then choose your .YML file.
8. Choose **Save & queue**.

## Example: Hello world

To see a more interesting example in action, replace the content in **.vsts-ci.yml** with this content:

```
steps:  
  
- script: |  
  echo hello world from %MyName%  
  echo Agent.HomeDirectory is %CD%  
  workingDirectory: $(Agent.HomeDirectory)  
  env:  
    MyName: $(Agent.MachineName)  
    condition: and(succeeded(), eq(variables['agent.os'], 'windows_nt'))  
    displayName: Greeting from Windows machine  
  
- script: |  
  echo hello world from $MyName  
  echo Agent.HomeDirectory is $PWD  
  workingDirectory: $(Agent.HomeDirectory)  
  env:  
    MyName: $(Agent.MachineName)  
    condition: and(succeeded(), in(variables['agent.os'], 'darwin', 'linux'))  
    displayName: Greeting from macOS or Linux machine
```

Queue the build on any of our hosted agent pools, including **Hosted VS 2017**, **Hosted Linux Preview** or **Hosted macOS Preview**. You'll get different kinds of greetings from each of these types of agent.

## Look up tasks

To look up the syntax of the tasks that are built into VSTS and TFS, see <https://github.com/Microsoft/vsts-tasks/tree/master/Tasks>.

For example, if you want to use the **Copy Files** task, go to <https://github.com/Microsoft/vsts-tasks/tree/master/Tasks/CopyFiles> and then click the **task.json** file. In this file you'll find the name of task, which in this case is `CopyFiles`. You'll also find the valid `inputs`, for example the `SourceFolder` input.

## Learn more

To learn more about what you can do with YAML builds, see <https://github.com/Microsoft/vsts-agent/blob/master/docs/preview/yamlgettingstarted.md>.

# Build your Azure cloud service

9/27/2017 • 2 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

Here we'll show you how to define your continuous integration (CI) process for your Azure cloud service project.

## Get set up

For the instructions in this topic, you need an Azure cloud service project in Visual Studio.

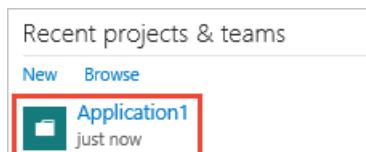
### TIP

If you don't yet have an app but want to try this out, then see the [Q&A below](#).

## Define your CI build process

### Create the build definition

#### 1. Open your team project in your web browser ▾

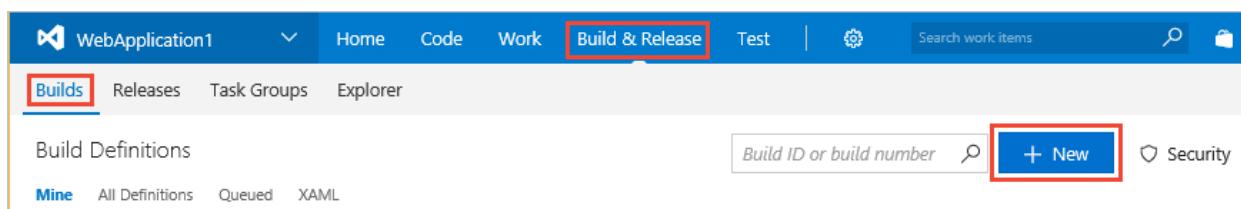


(If you don't see your team project listed on the home page, select **Browse**.)

- On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_team_project}`
- VSTS: `https://{your_account}.visualstudio.com/DefaultCollection/{your_team_project}`

The TFS URL doesn't work for me. How can I get the correct URL?

#### 2. Create a build definition (Build and Release tab > Builds) ▾



3. Select the **Azure Cloud Service** template.
4. As the repository source, select the team project, repository, and branch.
5. Remove the **Azure Cloud Service Deployment task** from the build definition, since you will be deploying the cloud service later through a release definition.

### Enable continuous integration (CI)

On the Triggers tab, enable **continuous integration** (CI). This tells the system to queue a build whenever someone on your team commits or checks in new code.

## Queue and test the build

Save the build definition and queue a new build by selecting the **Queue new build** command. Once the build is

done, click **Artifacts** and then **Explore** to see the cloud service package (.cspkg file) produced by the build. This is the package that your release definition will consume to deploy your app.

## Deploy your app

After you've run the build, you're ready to create a release definition to deploy your app to:

-  An Azure cloud service

## Q&A

### How do I create an Azure cloud service solution?

1. In Visual Studio, [connect to your team project](#).
2. On the Team Explorer home page (Keyboard: Ctrl + 0, H), under **Solutions**, click **New**.
3. Select the **Cloud** templates section, and then choose the **Azure Cloud Service** template.
4. When prompted for the roles in the cloud service, choose the **ASP.NET Web role** to the project.
5. When prompted for the type of ASP.NET project for the Web role, choose the **MVC** project.
6. [Commit and push \(Git\)](#) or [check in \(TFVC\)](#) your code.

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build your SQL server database

9/27/2017 • 1 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

Here we'll show you how to define your continuous integration (CI) process for your SQL server database project.

## Get set up

For the instructions in this topic, you need a SQL server database project in Visual Studio.

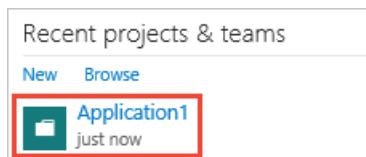
### TIP

If you don't yet have an app but want to try this out, then see the [Q&A below](#).

## Define your CI build process

### Create the build definition

#### 1. Open your team project in your web browser ▾

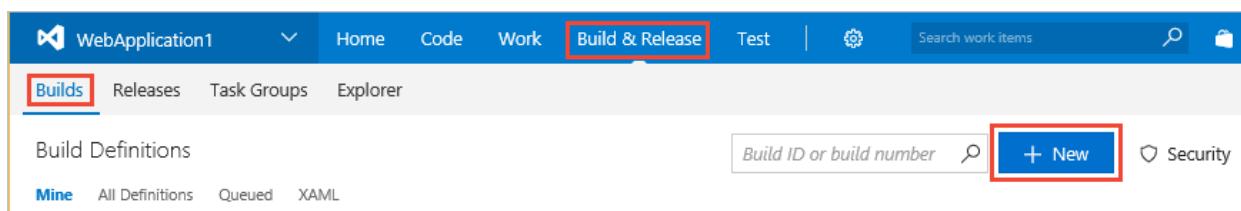


(If you don't see your team project listed on the home page, select **Browse**.)

- On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_team_project}`
- VSTS: `https://{your_account}.visualstudio.com/DefaultCollection/{your_team_project}`

The TFS URL doesn't work for me. How can I get the correct URL?

#### 2. Create a build definition (Build and Release tab > Builds) ▾



#### 3. Select the **Visual Studio** template.

#### 4. As the repository source, select the team project, repository, and branch.

### Enable continuous integration (CI)

On the Triggers tab, enable **continuous integration** (CI). This tells the system to queue a build whenever someone on your team commits or checks in new code.

## Queue and test the build

Save the build definition and queue a new build by selecting the **Queue new build** command. Once the build is done, click **Artifacts** and then **Explore** to see the DACPAC (.dacpac file) produced by the build. This is the package that your release definition will consume to deploy your database.

# Deploy your database

After you've run the build, you're ready to create a release definition to deploy your database to:

-  Azure SQL Server
-  SQL Server

## Q&A

### How do I create an SQL server database solution?

1. In Visual Studio, [connect to your team project](#).
2. On the Team Explorer home page (Keyboard: Ctrl + 0, H), under **Solutions**, click **New**.
3. Select the **SQL Server** templates section, and then choose the **SQL Server Database Project** template.
4. [Commit and push \(Git\)](#) or [check in \(TFVC\)](#) your code.

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Use a PowerShell script to customize your build process

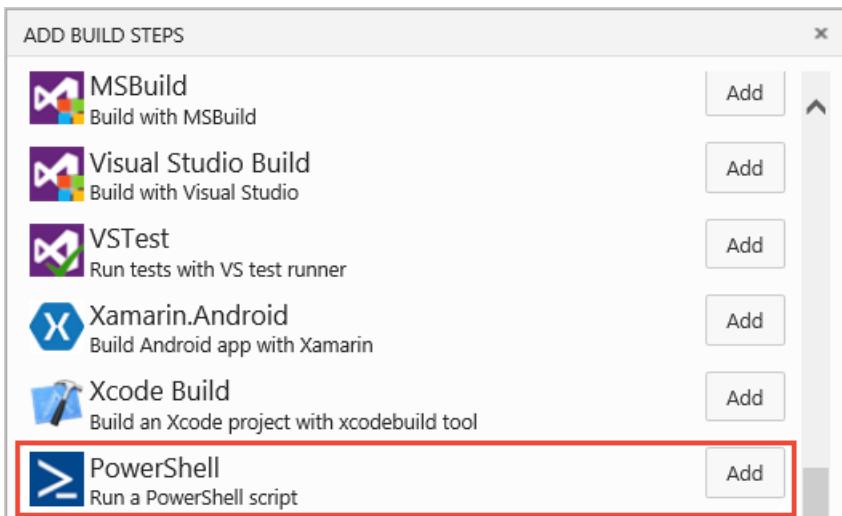
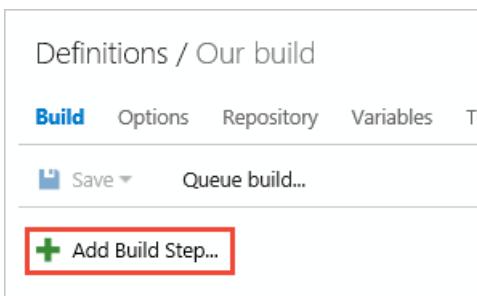
12/19/2017 • 3 min to read • [Edit Online](#)

[VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions \(XAML builds\)](#)

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 process.

You can run a PowerShell Script on a [Windows build agent](#).

1. Push your script into your repo.
2. Add a PowerShell build step.



3. Drag the build step where you want it to run.

4. Specify the name of the script.

## Example: Version your assemblies

For example, to version your assemblies, copy and upload this script to your team project:

```
##-----  
## <copyright file="ApplyVersionToAssemblies.ps1">(c) Microsoft Corporation. This source is subject to the  
Microsoft Permissive License. See  
http://www.microsoft.com/resources/sharedsource/licensingbasics/sharedsourcelicenses.mspx. All other rights  
reserved.</copyright>  
##-----  
# Look for a 0.0.0.0 pattern in the build number.  
# To find more information about the build number, see  
# https://msdn.microsoft.com/en-us/library/hh128802.aspx
```

```

# It's good to use it to version the assemblies.
#
# For example, if the 'Build number format' build process parameter
# $(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.r)
# then your build numbers come out like this:
# "Build HelloWorld_2013.07.19.1"
# This script would then apply version 2013.07.19.1 to your assemblies.

# Enable -Verbose option
[CmdletBinding()]

# Regular expression pattern to find the version in the build number
# and then apply it to the assemblies
$VersionRegex = "\d+\.\d+\.\d+\.\d+"

# If this script is not running on a build server, remind user to
# set environment variables so that this script can be debugged
if(-not ($Env:BUILD_SOURCESDIRECTORY -and $Env:BUILDDATE))
{
    Write-Error "You must set the following environment variables"
    Write-Error "to test this script interactively."
    Write-Host '$Env:BUILD_SOURCESDIRECTORY - For example, enter something like:'
    Write-Host '$Env:BUILD_SOURCESDIRECTORY = "C:\code\FabrikamTFVC\HelloWorld"'
    Write-Host '$Env:BUILDDATE - For example, enter something like:'
    Write-Host '$Env:BUILDDATE = "Build HelloWorld_0000.00.00.0"'
    exit 1
}

# Make sure path to source code directory is available
if (-not $Env:BUILD_SOURCESDIRECTORY)
{
    Write-Error ("BUILD_SOURCESDIRECTORY environment variable is missing.")
    exit 1
}
elseif (-not (Test-Path $Env:BUILD_SOURCESDIRECTORY))
{
    Write-Error "BUILD_SOURCESDIRECTORY does not exist: $Env:BUILD_SOURCESDIRECTORY"
    exit 1
}
Write-Verbose "BUILD_SOURCESDIRECTORY: $Env:BUILD_SOURCESDIRECTORY"

# Make sure there is a build number
if (-not $Env:BUILDDATE)
{
    Write-Error ("BUILDDATE environment variable is missing.")
    exit 1
}
Write-Verbose "BUILDDATE: $Env:BUILDDATE"

# Get and validate the version data
$VersionData = [regex]::matches($Env:BUILDDATE,$VersionRegex)
switch($VersionData.Count)
{
    0
    {
        Write-Error "Could not find version number data in BUILDDATE."
        exit 1
    }
    1 {}
    default
    {
        Write-Warning "Found more than one instance of version data in BUILDDATE."
        Write-Warning "Will assume first instance is version."
    }
}
$NewVersion = $VersionData[0]
Write-Verbose "Version: $NewVersion"

# Apply the version to the assembly property files

```

```

$files = gci $Env:BUILD_SOURCESDIRECTORY -recurse -include "*Properties*","My Project" |
?{ $_.PSIsContainer } |
foreach { gci -Path $_.FullName -Recurse -include AssemblyInfo.* }
if($files)
{
    Write-Verbose "Will apply $NewVersion to $($files.count) files."

    foreach ($file in $files) {
        $filecontent = Get-Content($file)
        attrib $file -r
        $filecontent -replace $VersionRegex, $NewVersion | Out-File $file
        Write-Verbose "$file.FullName - version applied"
    }
}
else
{
    Write-Warning "Found no files."
}

```

Add the build step to your build definition.

Definitions / Our build

**Build** Options Repository Variables Triggers General Retention History

Save Queue build...

**Add Build Step...**

**PowerShell: Scripts/ApplyVersionToAssemblies.ps1**

Enabled

Script filename  ... ⓘ

Arguments  ⓘ

Advanced

Visual Studio Build  
Build solution \*\*\\*.sln

VSTest  
Test Assemblies \*\*\\*test\*.dll;-\*\obj\\*\*

Specify your build number with something like this:

Definitions / Our build

Build Options Repository Variables Triggers **General** Retention History

Save Queue build...

Default queue  ⏺

Description

Build number format  ⏺

\$(BuildDefinitionName)\_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$Rev:.r

## Use the OAuth token to access the REST API

To enable your script to use the build process OAuth token, go to the **Options** tab of the build definition and select **Allow Scripts to Access OAuth Token**.

After you've done that, your script can use to SYSTEM\_ACCESTOKEN environment variable to access the [VSTS](#)

[REST API](#). For example:

```
$url = "$($env:SYSTEM_TEAMFOUNDATIONCOLLECTIONURI)$env:SYSTEM_TEAMPROJECTID/_apis/build-release/definitions/$($env:SYSTEM_DEFINITIONID)?api-version=2.0"
Write-Host "URL: $url"
$definition = Invoke-RestMethod -Uri $url -Headers @{
    Authorization = "Bearer $env:SYSTEM_ACCESSTOKEN"
}
Write-Host "Definition = $($definition | ConvertTo-Json -Depth 1000)"
```

## Q&A

**What variables are available for me to use in my scripts?**

[Use variables](#)

**How do I set a variable so that it can be read by subsequent scripts and tasks?**

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

**Which branch of the script does the build run?**

The build runs the script same branch of the code you are building.

**What kinds of parameters can I use?**

You can use named parameters. Other kinds of parameters, such as switch parameters, are not yet supported and will cause errors.

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) 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

9/13/2017 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

For some workflows you need your build process to run Git commands. For example, after a CI build on a feature branch is done, the team might want to merge the branch to master.

Git.exe is available on [hosted agents](#) and on [on-premises agents](#).

## Enable scripts to run Git commands

### Grant version control permissions to the build service

Go to the [Version Control control panel tab](#) □

- VSTS: [https://{your-account}.visualstudio.com/DefaultCollection/{your-team-project}/\\_admin/\\_versioncontrol](https://{your-account}.visualstudio.com/DefaultCollection/{your-team-project}/_admin/_versioncontrol)
- On-premises: [https://{your-server}:8080/tfs/DefaultCollection/{your-team-project}/\\_admin/\\_versioncontrol](https://{your-server}:8080/tfs/DefaultCollection/{your-team-project}/_admin/_versioncontrol)



If you see this page, select the repo, and then click the link:

The screenshot shows the 'Control panel > fabrikam' interface. The top navigation bar has tabs: Overview (highlighted with a red box), Settings, Security, Process, Build, Agent pools, and Extensions\*. Below this, the 'Account / fabrikam' section displays a 'Description' field and a 'New team project' button. The 'Projects' section lists two repositories: 'Fabrikam' and 'OurProject' (highlighted with a red box). Both repositories have their names, icons, and 'Process' listed as 'Agile'.

The screenshot shows the 'Control panel > fabrikam > OurProject' interface. The top navigation bar has tabs: Overview, Work, Security, Alerts, and Version Control (highlighted with a red box).

On the **Version Control** tab, select the repository in which you want to run Git commands, and then select **Project Collection Build Service**.

The screenshot shows the 'Security for all Git repositories' interface in VSTS. On the left, there's a sidebar with 'Repositories' and 'New repository' options, followed by a list of 'Git repositories': OurProject, Portal-Client, Portal-Server, Store-Client, and Store-Server. The 'Git repositories' section is expanded. On the right, there's a large list of 'VSTS Groups' under the heading 'Security for all Git repositories'. The groups listed are: Project Collection Administrators, Project Collection Build Service Accounts, Project Collection Service Accounts, Build Administrators, Contributors, Project Administrators, Readers, and Project Collection Build Service (Application-...). The last group, 'Project Collection Build Service (Application-...)', is highlighted with a red border around its icon and name.

Grant permissions needed for the Git commands you want to run. Typically you'll want to grant:

- **Branch creation:** Allow
- **Contribute:** Allow
- **Read:** Inherited allow
- **Tag creation:** Inherited allow

When you're done granting the permissions, make sure to click **Save changes**.

#### Enable your build definition to run Git.exe

On the [variables tab](#) set this variable:

NAME	VALUE
system.prefergit	true

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 process. So make sure to:

- Delete local branches you create.
- Undo git config changes.

If you run into problems using an on-premises agent, to make sure the repo is clean:

- 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.exe](#).

On the [build tab](#) add this step:

TASK	ARGUMENTS
 <a href="#">Utility: Command Line</a> List the files in the Git repo.	<b>Tool:</b> <code>git</code> <b>Arguments:</b> <code>ls-files</code>

### Merge a feature branch to master

You want a CI build to merge to master if the build succeeds.

Make sure to follow the above steps to [enable Git.exe](#).

On the [Triggers tab](#) select **Continuous integration (CI)** and include the branches you want to build.

Create `merge.bat` at the root of your repo:

```
@echo off
ECHO SOURCE BRANCH IS %BUILD_SOURCEBRANCH%
IF %BUILD_SOURCEBRANCH% == refs/heads/master (
    ECHO Building master branch so no merge is needed.
    EXIT
)
SET sourceBranch=origin/%BUILD_SOURCEBRANCH:refs/heads/=%
ECHO GIT CHECKOUT MASTER
git checkout master
ECHO GIT STATUS
git status
ECHO GIT MERGE
git merge %sourceBranch% -m "Merge to master"
ECHO GIT STATUS
git status
ECHO GIT PUSH
git push origin
ECHO GIT STATUS
git status
```

On the [build tab](#) add this as the last step:

TASK	ARGUMENTS
 <a href="#">Utility: Batch Script</a> Run merge.bat.	<b>Path:</b> <code>merge.bat</code>

## Q&A

**Can I run Git commands if my remote repo is in GitHub or an external Git service such as Bitbucket?**

Yes

**Which steps can I use to run Git commands?**

[Batch Script](#)

[Command Line](#)

[PowerShell](#)

[Shell Script](#)

## **How do I avoid triggering a CI build when the script pushes?**

Add `***NO_CI***` to your commit message. For example,

```
git merge origin/features/hello-world -m "Merge to master ***NO_CI***"
```

## **How does enabling scripts to run Git commands affect how the build process gets build sources?**

When you set `system.usegit` to `true`, the build process uses Git.exe instead of LibGit2Sharp to clone or fetch the source files.

## **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

## **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

## **I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) 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

1/26/2018 • 8 min to read • [Edit Online](#)

**VSTS | 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](#)

## 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 definition from the Android or Xamarin.Android build template. Or, if you already have a build definition, 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"/>

+ 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 (v) (refresh) (gear)

**Keystore Password** (i)

\$(keystore-password)

**Alias** (i)

\$(key-alias)

**Key Password** (i)

\$(key-password)

Save your build definition, 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.

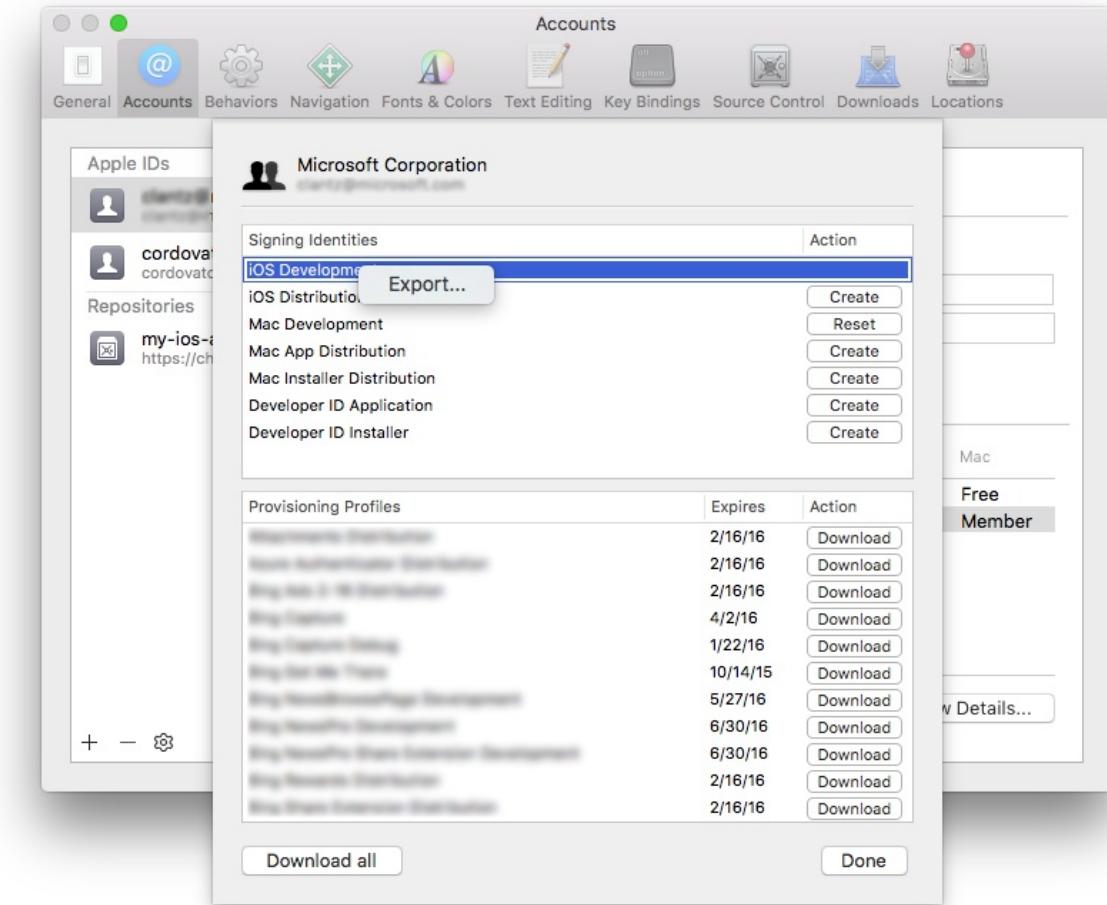
## Sign your Apple iOS, macOS, tvOS, or watchOS app

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.

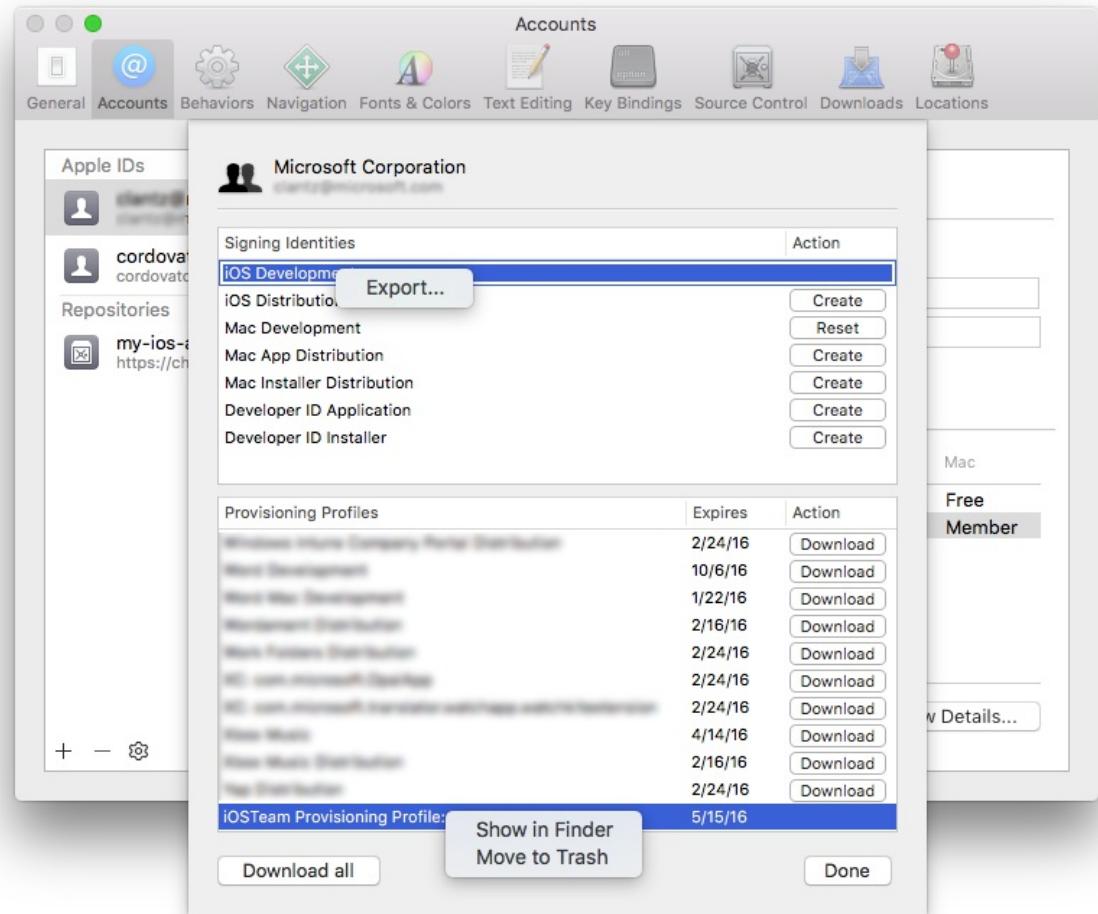


4. Alternatively, follow a similar process using the **Keychain Access** app on macOS or generate a signing certificate on Windows. Use the procedure [described in this article](#) if you prefer this method.

### 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

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)`

### Install the provisioning profile during your build

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**.

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.

#### Reference the files in your Xcode task

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.

#### Reference the files in your Xamarin.iOS task

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.

Save your build definition, and you are all set! The build agent will now be able to securely sign and provision your app.

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Deploy to Azure web app

1/19/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017.2

We'll show you how to set up continuous deployment of your ASP.NET or Node app to an Azure web app using Visual Studio Team Services (VSTS) or Microsoft Team Foundation Server (TFS). You can use the steps in this quickstart as long as your continuous integration process 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 app with Gulp](#)

You'll also need an Azure web app where you will deploy the app. If you don't have one already, create one now. If you need help, follow the steps in [this example](#).

## Define your CD release process

Your CD release process 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 definition:

- 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 definition that's automatically linked to the build definition.

The screenshot shows the VSTS Build & Release hub. At the top, it displays "SampleApp / Build 20170815.1". Below this are several buttons: "Edit build definition", "Queue new build...", "Download all logs as zip", "Retain indefinitely", and a red-bordered "Release" button. A green bar at the bottom indicates "Build succeeded". Below the bar, it says "Build 20170815.1" and "Ran for 2.6 minutes (Hosted VS2017), completed 10 days ago".

- Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**.

2. The easiest way to create a release definition is to use a template. If you are deploying a Node 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 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 definition from a build summary, check that the build definition and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release definition from the **Releases** tab, choose the **+ Add** link and select your build artifact.

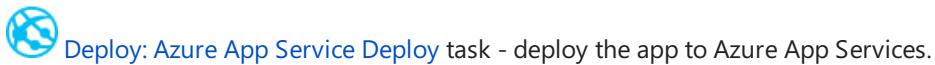
4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and the **master** branch is selected. If not, set these options now.

All definitions > **New Release Definition**

**Pipeline** Tasks Variables Retention Options History

Continuous deployment is not enabled by default when you create a new release definition from the **Releases** tab.

5. Open the **Task** tab, select the **Deploy Azure App Service** task, and configure it 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 VSTS and if you see an **Authorize** button next to the input, click on it to authorize VSTS 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 endpoint](#) to manually set up the connection.

MyFirstProject-CI - CD

Save + Release View releases Edit (Old editor) ...

Pipeline Tasks Variables History

- **App Service Name:** Select the name of the web app from your subscription.

6. Save the release definition.

## 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 definition with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose **+ Release** and select **Create Release**.

2. In the **Create 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. Open the **Logs** tab to watch the release console output.
5. After the release is complete, navigate to your site running in Azure using the Web App URL  
`http://{web_app_name}.azurewebsites.net`, and verify its contents.

## Next steps

- [Deploy to a staging slot and then swap to production](#)
- [Deploy multiple apps in the same release](#)
- [Apply environment-specific configurations](#)
- [Deploy to a Government cloud instead of a public cloud](#)

# How To: Extend your deployments to Azure App Services

1/19/2018 • 3 min to read • [Edit Online](#)

You can quickly and easily deploy your ASP.NET or Node app to an Azure App Services website using Visual Studio Team Services (VSTS) or Microsoft Team Foundation Server (TFS) 2017.2, 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:

- [Deploy to a staging slot and then swap to production](#)
- [Deploy multiple apps in the same release](#)
- [Apply environment-specific configurations](#)
- [Deploy to a Government cloud instead of a public cloud](#)

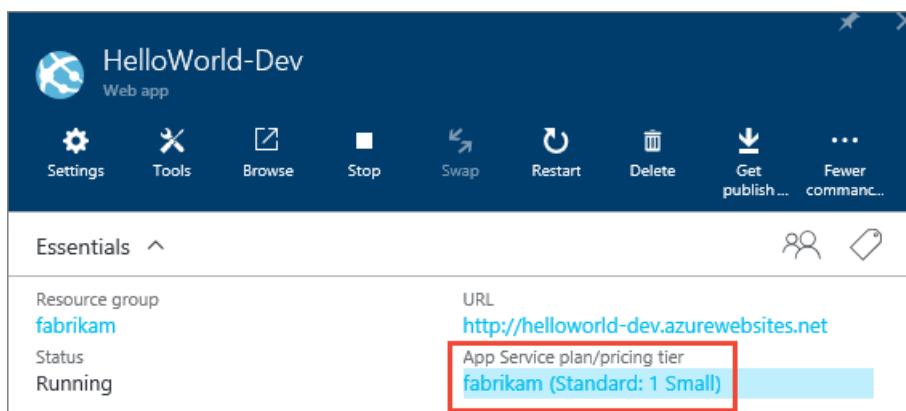
## Prerequisites

You should have worked through the example [CD to Azure App Services](#) before you attempt any of these steps. This ensures that you have the release definition, build artifacts, and website required.

## Deploy to a staging slot and then swap to production

If you want to deploy to a staging slot in an Azure App Services website, and then swap the staging and production slots, perform these steps.

1. Ensure that your Azure App Services plan is a standard or premium level. This is required to configure deployment slots.



2. Configure a **Staging** slot in the Azure App Services website.

NAME	STATUS
helloworld-dev-staging	Running

You do not need to configure a **Production** slot because it is implicitly present in all Azure App Services websites.

For more information on configuring deployment slots, see [Set up staging environments for web apps in Azure App Service](#) on the Azure website.

### 3. Configure the tasks in the environment of the release definition as follows:

**Deploy: Azure App Service Deploy** - Deploy the app to Azure App Services.

- **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 endpoint](#).
- **App Service name:** Select your App Service.
- **Deploy to slot:** .
- **Slot:**

**Deploy: Azure App Service Manage** - Swap slots.

- **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 endpoint](#).
- **Action:**
- **App Service name:** Select your App Service.
- **Resource group:** Select the resource group to which your App Service belongs.
- **Source slot:**

You might also consider [Configuring Auto Swap](#) for your Azure App Services web app to automatically swap the app into production after successful deployment.

## Deploy multiple apps in the same release

If your CI build process builds multiple apps, you can deploy them in the same release. To do this, configure a separate instance of the **Azure App Service Deploy** task for each app. In each instance of the task:

- Specify a unique value for **App Service Name**

- Specify the **Web Deploy Package** with a reference to a specific app. For example, replace the default value with a more specific value such as `$(build.stagingDirectory)\**\WebApplication1.zip`

## Apply environment-specific configurations

If you deploy releases to multiple environments, you can substitute configuration settings in **Web.config** and other configuration files of your website using these steps:

- Define environment-specific configuration settings in the **Variables** tab of an environment in a release definition; for example, `connectionString = <value>`.
- In the **Azure App Service Deploy** task, select the check box for **XML variable substitution** under **File Transforms and Variable Substitution Options**. This option is present only in version 3.\* and above of the task. It is not yet available in TFS.

If you prefer to manage environment configuration settings in your own database or Azure keyvault, add a task to the environment to read and emit those values using

```
##vso[task.setvariable variable=connectionString;issecret=true]<value>
```

For more details, see [Managing Configuration & App Settings for Multiple Environments in Your CD Pipeline](#).

## Deploy to a Government cloud instead of a public cloud

Do this by creating a suitable service endpoint for your Azure Government Cloud subscription. See [Azure Government Cloud deployments](#) for details.

# How To: Extend your deployments to IIS Deployment Groups

1/19/2018 • 2 min to read • [Edit Online](#)

You can quickly and easily deploy your ASP.NET or Node app to an IIS Deployment Group using Visual Studio Team Services (VSTS) or Microsoft 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 environment-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 definition, 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 environment-specific configurations

If you deploy releases to multiple environments, you can substitute configuration settings in **Web.config** and other configuration files of your website using these steps:

1. Define environment-specific configuration settings in the **Variables** tab of an environment in a release definition; for example, `<connectionStringKeyName> = <value>`.
2. In the **IIS Web App Deploy** task, select the checkbox for **XML variable substitution** under **File Transforms and Variable Substitution Options**.

If you prefer to manage environment configuration settings in your own database or Azure keyvault, add a task to the environment 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 phase** and use the slider to configure the **Maximum number of targets in parallel**.

The screenshot shows the 'Tasks' tab selected in the Azure DevOps pipeline interface. On the left, there's a list of tasks: 'Prod' (Deployment group phase), 'Deployment group phase' (selected and highlighted with a red box), 'Manage IISWebsite', and 'Deploy IIS Website/App:'. The right side shows configuration for the selected task. It includes fields for 'Display name' (Deployment group phase), 'Deployment targets' (with a dropdown for 'Deployment group' set to 'SampleGroup'), 'Required tags', and deployment settings. A red box highlights the 'Targets to deploy to in parallel' section, which contains a radio button for 'Multiple' (selected) and a slider for 'Maximum number of targets in parallel' set to 50% (3 targets). There's also a 'Timeout' field.

## 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 phases to environments in the release definition, and a task in each phase as follows:

**First Run on deployment group phase** for configuration of web servers.

- **Deployment group:** Select the deployment group you created in the [previous example](#).
- **Machine tags:** `web`

Then add an **IIS Web App Deploy** task to this phase.

**Second Run on deployment group phase** for configuration of database servers.

- **Deployment group:** Select the deployment group you created in the [previous example](#).
- **Machine tags:** `database`

Then add a **SQL Server Database Deploy** task to this phase.

# Deploy your ASP.NET app to an Azure cloud service

1/19/2018 • 3 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Continuous deployment means starting an automated deployment process whenever a new successful build is available. Here we'll show you how to set up continuous deployment of your ASP.NET app to an Azure cloud service using Release Management.

You can also use these steps to deploy your app to an Azure Government Cloud by creating a suitable service endpoint for your Azure Government Cloud subscription. For more details, see the [Azure Government Cloud deployments](#).

## Get set up

### Begin with a CI build

Before you begin, you'll need a CI build that publishes your Cloud Service package. To set up CI, see:

- [Build your Azure cloud service](#)

### Azure storage

An Azure blob storage container is required for deploying to Azure cloud services. Carry out the following steps in the Azure portal to create one.

1. Sign into the Azure management portal and choose the **+New** icon in the left panel, then choose **Data + Storage**. Select **Storage Account** from the list.
2. At the bottom of the **Storage Account** blade, in the **Select a deployment model** list, choose **Classic** and then choose **Create**.
3. In the **Create Storage Account** blade:
  - Enter a name for the new storage account.
  - Select an existing Resource Group, or create a new one.
  - Select a location for the new storage account.
  - Leave all the other settings at their default values, and choose **Create**.
4. After the storage account has been created, open its blade and choose the **Blobs** tile. In the **Blob service** blade, choose the **Containers** tile and, in the **Container** blade, choose the **+Container** icon at the top to create a new container.
5. In the **New Container** blade, type a name for the container. Select **Container** in the **Access type** list, and choose **Create**.

## Define and test your CD release process

Continuous deployment (CD) means starting an automated release process whenever a new successful build is available. Your CD release process picks up the artifacts published by your CI build and then deploys them to your Azure cloud service.

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 definition that's automatically linked to the build definition.

- Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**.
2. Select the **Azure Cloud Service Deployment** template and click **Next**.
  3. In the **Artifacts** section, make sure your CI build definition that creates the cloud services artifacts is selected.
  4. Select the **Continuous deployment** check box, and then choose **Create**.
  5. Select the **Azure Cloud Service Deployment** task and configure it as follows:

 Deploy: Azure Cloud Service Deployment - Deploy the app to an Azure cloud service.

- **Azure Subscription (Classic)**: Select an Azure Classic service endpoint. If you have not created one already, create one now by choosing **Add**. Then return to your release definition, refresh the **Azure Subscription** list, and select the connection you just created.
- **Storage account**: Select the storage account you created earlier.
- **Service name**: Select the name of an existing cloud service, or enter the name of a new cloud service.

If your Azure subscription is defined in an Azure Government Cloud, ensure your deployment process meets the relevant compliance requirements. For more details, see [Azure Government Cloud deployments](#).

6. Edit the name of the release definition, click **Save**, and click **OK**. Note that the default environment is named **Environment1**, which you can edit by clicking directly on the name.

You're now ready to create a release, which means to start the process of running the release definition with the artifacts produced by a specific build. This will result in deploying the build to Azure:

1. Choose + **Release** and select **Create Release**.
2. Select the build you just completed in the highlighted drop-down list and choose **Create**.
3. Choose the release link in the popup message. For example: "Release **Release-1** has been created".
4. Open the **Logs** tab to watch the release console output.
5. After the release is complete, navigate to your site running in Azure cloud services and verify its contents.

## Q&A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deployments to Azure Government Cloud

1/19/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2017

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.

Visual Studio Team Services (VSTS) 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 a [service endpoint](#) connection using either the [Azure Classic service endpoint](#) or the [Azure Resource Manager service endpoint](#).

**Before you configure a service endpoint, you must ensure you meet all relevant compliance requirements for your application.**

### Azure Classic service endpoint

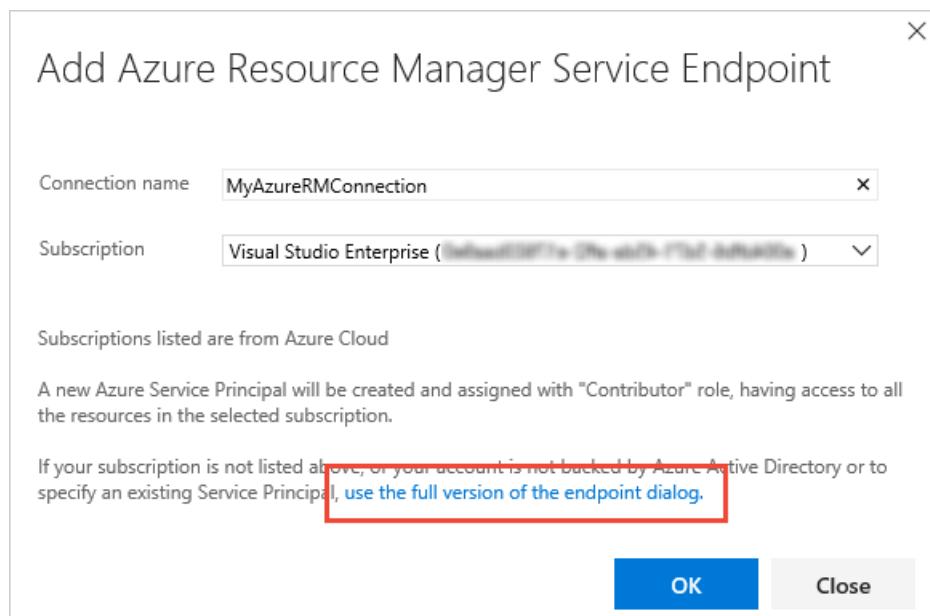
Defines and secures a connection to a Microsoft Azure subscription using Azure credentials or an Azure management certificate.

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 endpoint in task properties. This is not the name of your Azure account or subscription.
Environment	Required. Select the Azure Government Cloud where your target Azure 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.

## Azure Resource Manager service endpoint

Defines and secures a connection to a Microsoft Azure subscription using Service Principal Authentication (SPA).

You must use the full version of the dialog when connecting to an Azure Government Cloud. Choose the link near the bottom of the dialog.



PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your Azure account or subscription.
Environment	Required. Select the Azure Government Cloud where your target Azure subscription is defined.
Subscription ID	Required only if you want to use an existing service principal. The GUID-like identifier for your Azure subscription (not the subscription name).
Subscription Name	Required only if you want to use an existing service principal. The name of your Microsoft Azure subscription (account).
Service Principal ID	Required only if you want to use an existing service principal. The Azure Active Directory client ID of the account.
Service Principal Key	Required only if you want to use an existing service principal. The Azure Active Directory client key of the account.
Tenant ID	Required only if you want to use an existing service principal. The ID of the client tenant in Azure Active Directory.

Follow these steps to configure the service endpoint parameters:

1. Download and run [this PowerShell script](#) in an Azure PowerShell window. When prompted, enter your subscription name, password, role (optional), and the Azure Government Cloud name.
2. Switch from the simplified version of the dialog to the full version using the link in the dialog.
3. Enter a user-friendly name to use when referring to this service endpoint connection.
4. Select the Environment name (such as Azure Cloud or an Azure Government Cloud).
5. Copy these fields from the output of the PowerShell script into the Azure subscription dialog textboxes:

- Subscription ID
- Subscription Name
- Service Principal ID
- Service Principal Key
- Tenant ID

See [this blog post](#) for details about using service principal authentication.

See also [Troubleshoot Azure Resource Manager service endpoints](#).

## Next

- [Deploy an Azure web app](#)
- [Deploy an Azure cloud service](#)
- [Examples index](#)

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Implement continuous deployment of your app using Kubernetes to Azure Container Service

2/6/2018 • 6 min to read • [Edit Online](#)

Continuous deployment (CD) means starting an automated deployment process whenever a new successful build is available. Here, we'll show you how to set up continuous delivery of a Docker based app by using VSTS to a Kubernetes cluster running in Azure Container Service.

Azure Container Service (ACS) allows to deploy and manage containers using Kubernetes, Docker Swarm, Mesosphere DC/OS orchestrators. You can deploy these three orchestrators on Azure by using the portal, an Azure Resource Manager template, or the Azure CLI.

The [Azure Container Registry \(ACR\)](#) is an implementation of the open source Docker Registry. ACR is now available as an Azure Service and is fully compatible with all three orchestrators. ACR is used as a private registry to store Docker images for enterprise applications, instead of needing to use the public Docker Hub.

## Get set up

### Begin with a CI build

Before you begin, you'll need a CI build that publishes your app. To set up CI for your specific type of app, see:

- [Build and deploy your app](#)

### Modify your build to create a Docker container

Next, you'll need to modify your build definition to create a Docker container for deployment using ACS.

1. Add two instances of the **Docker** task to the end of your build definition.
2. Add one instance of the **Publish Build Artifacts** task to the end of your build definition.
3. Configure the tasks as follows:



**Build: Docker** Build the container image from the Docker file.

- **Container Registry type:** `Azure Container Registry`
- **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 endpoint](#).
- **Azure Container Registry:** Select the target Azure container registry.
- **Action:** `Build an image`
- **Image name:** Enter the name for your Docker image.
- **Qualify Image Name:** Checked
- **Additional Image Tags:** `$(Build.BuildId)`



**Build: Docker** Push the container image to a container registry.

- **Container Registry type:** Azure Container Registry
- **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 endpoint](#).
- **Azure Container Registry:** Select the target Azure container registry.
- **Action:** Push an image
- **Image name:** Enter the name of your Docker image.
- **Qualify Image Name:** Checked
- **Additional Image Tags:** \$(Build.BuildId)

 **Build: Publish Build Artifacts** - Publish the Kubernetes configuration files used for creating the [deployment](#) and [service](#) in the cluster. These files are added to the [repository](#).

- **Path to Publish:** Path to the artifacts you want to publish. For example, k8config
- **Artifact name:** yaml
- **Artifact Type:** Server

4. Save your build definition.

## Define and test your CD release process

Continuous deployment (CD) means starting an automated release process whenever a new successful build is available. Your CD release process picks up the artifacts published by your CI build and then deploys them.

1. Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**
2. Select the **Deploy to Kubernetes cluster** template and choose **Next**.
3. In the **Artifacts** section, make sure your CI build definition for the Docker package is selected as the artifact source.
4. Select the **Continuous deployment** check box, and then choose **Create**.
5. Add two more instances of the **Deploy to Kubernetes** task to your release definition. This task uses the `kubectl` command to execute operations on a Kubernetes cluster.
6. Configure the tasks as follows:



**Deploy: Deploy to Kubernetes** - Create the deployment and secret.

- **Container Registry type:** Azure Container registry
- **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 endpoint](#).
- **Azure Container registry:** Select the Azure container registry to which you pushed your container images.
- **Secret name:** Name of the Docker registry secret. You can use this secret name in the Kubernetes YAML configuration file.

- **Command:** `apply` (you can run any kubectl command)
- **Use Configuration file:** Checked
- **Configuration file:** Select the **deployment.yaml** file that was published as an artifact from the build. Example: `$(System.DefaultWorkingDirectory)/Kubernetes-ACS-CI/yaml/deployment.yaml`



**Deploy: Deploy to Kubernetes** - Create the service using the yaml file.

- **Container Registry type:** `Azure Container registry`
- **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 endpoint](#).
- **Azure Container registry:** Select the Azure container registry to which you pushed your container images.
- **Secret name:** Name of the Docker registry secret. You can use this secret name in the Kubernetes YAML configuration file.
- **Command:** `apply`
- **Use Configuration file:** Checked
- **Configuration file:** Select the **service.yaml** file that was published as an artifact from the build. Example: `$(System.DefaultWorkingDirectory)/Kubernetes-ACS-CI/yaml/service.yaml`



**Deploy: Deploy to Kubernetes**

Update with the latest image.

- **Container Registry type:** `Azure Container registry`
- **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 endpoint](#).
- **Azure Container registry:** Select the Azure container registry to which you pushed your container images.
- **Secret name:** Name of the Docker registry secret. You can use this secret name in the Kubernetes YAML configuration file.
- **Command:** `set`
- **Arguments:** Arguments to pass to teh command. For example, `image deployment/coreserverdeployment core-server=image:tag` where you are using a private registry (so the image name must be prefixed with the container registry name). We also use the Build Id to tag our images here, so the `image:tag` value will be `{your-acr-name}.azurecr.io/docker-dotnetcore:$(Build.BuildId)`. `docker-dotnetcore` is the image name used in the build.

7. Edit the name of the release definition, choose **Save**, and choose **OK**. Note that the default environment is named Environment1, which you can edit by clicking directly on the name.

You're now ready to create a release, which means to start the process of running the release definition with the artifacts produced by a specific build. This will result in deploying the build to Azure:

1. Choose **+ Release** and select **Create Release**.

2. Select the build you just completed in the highlighted drop-down list and choose **Create**.
3. Choose the release link in the popup message. For example: "Release **Release-1** has been created".
4. Open the **Logs** tab to watch the release console output.

## Q&A

### Why use the Build ID as the tag when deploying?

In this example, we used the `kubectl set image` command with the Build ID as the tag. Using the Build ID as the tag has an added advantage of traceability. Avoid using the latest tag with the container image. An alternate approach is to modify the YAML file with the Build ID used to tag the image.

### How do I define separate environments for my release?

The [Kubernetes namespace](#) allows complete separation of resources and management within the same cluster. So namespace can be used to create multiple environments such as *Dev*, *QA*, and *Production* in the same ACS Kubernetes cluster.

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy your Web Deploy package to IIS servers using WinRM

1/19/2018 • 6 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

A simpler way to deploy web applications to IIS servers is by using [deployment groups](#) instead of WinRM. Deployment groups are currently in preview for some accounts in VSTS. They are not yet available in TFS.

Continuous deployment means starting an automated deployment process whenever a new successful build is available. Here we'll show you how to set up continuous deployment of your ASP.NET or Node app to one or more IIS servers using Release Management. 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 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
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.

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 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 to your VSTS account or TFS.

## Define and test your CD release process

Continuous deployment (CD) means starting an automated release process whenever a new successful build is available. Your CD release process 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 definition that's automatically linked to the build definition.
- Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**.

2. Select the **Empty** template and click **Next**.

3. In the **Artifacts** section, make sure your CI build definition that publishes the Web Deploy package is selected as the artifact source.

4. Select the **Continuous deployment** check box, and then click **Create**.

5. On the **Variables** tab of the environment in release definition, configure a variable named **WebServers** with the list of IIS servers as its value; for example `machine1,machine2,machine3`.

6. Configure the following tasks in the environment:



[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 step.

- **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 definition, click **Save**, and click **OK**. Note that the default environment is named `Environment1`, which you can edit by clicking directly on the name.

You're now ready to create a release, which means to start the process of running the release definition with the artifacts produced by a specific build. This will result in deploying the build to IIS servers:

1. Choose **+ Release** and select **Create Release**.
2. Select the build you just completed in the highlighted drop-down list and choose **Create**.
3. Choose the release link in the popup message. For example: "Release **Release-1** has been created".
4. Open the **Logs** tab to watch the release console output.
5. After the release is complete, navigate to your site running on the IIS servers, and verify its contents.

## Q&A

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy your database to Azure SQL Database using DACPACs

1/19/2018 • 3 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Continuous deployment means starting an automated deployment process whenever a new successful build is available. Here we'll show you how to set up continuous deployment of your database packaged as a DACPAC to an Azure SQL Database using Release Management.

## Get set up

### Begin with a CI build

Before you begin, you need a CI build that publishes your SQL server package. To set up CI, see:

- [Continuous integration for SQL Server databases](#)

### Azure SQL Database

Carry out the following steps to set up an Azure SQL Database server to which the DACPAC of the database will be deployed.

1. Sign into the Azure management portal and choose the **+New** icon in the left panel, then choose **Data + Storage**. Select **SQL Database** from the list.
2. In the **SQL Database** blade, enter a name for Azure SQL Database and then choose **Server** to configure the required settings for the server.
3. In the **Server** blade, choose **Create a new server**.
4. In the **New server** blade, enter a name for the server and enter the admin login and password for the new server. Leave all other settings as they are and choose **OK**.
5. Back in the **SQL Database** blade, leave all the other settings at their default values and choose **Create**.
6. After the Azure SQL Database server and database have been created, open its blade and make a note of the **Server name**.

## Define and test your CD release process

Continuous deployment (CD) means starting an automated release process whenever a new successful build is available. Your CD release process picks up the artifacts published by your CI build and then deploys them to your database.

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 definition that's automatically linked to the build definition.
  - Open the **Releases** tab of the **Build & Release** hub, open the **+** drop-down in the list of release definitions, and choose **Create release definition**.
2. Select the **Empty** template and choose **Next**.

3. In the **Artifacts** section, make sure your CI build definition that publishes the DACPAC is selected as the artifact source.
4. Select the **Continuous deployment** check box, and then choose **Create**.
5. Add a **SQL Database** task to the default environment and configure it as follows:



Deploy: Azure SQL Database Deployment - Deploy the database to Azure SQL Database.

- **Azure Connection Type:** .
- **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 endpoint](#).
- **Azure SQL Server Name:** Enter the name of the SQL Database server you created earlier.
- **Database Name:** Enter the name of database.
- **Server Admin Login:** Enter the admin user name for your SQL Database.
- **Password:** Enter the admin password for your SQL Database. To hide the password, create a variable for it in the environment.
- **Firewall - Specify Firewall Rules Using:**

[How can I perform other actions on a SQL Server or Azure SQL Database?](#)

6. Edit the name of the release definition, choose **Save**, and choose **OK**. Note that the default environment is named Environment1, which you can edit by clicking directly on the name.

You're now ready to create a release, which means to start the process of running the release definition with the artifacts produced by a specific build. This will result in deploying the database DACPAC to Azure SQL Database:

1. Choose **+ Release** and select **Create Release**.
2. Select the build you just completed in the highlighted drop-down list and choose **Create**.
3. Choose the release link in the popup message. For example: "Release **Release-1** has been created".
4. Open the **Logs** tab to watch the release console output.

## Q&A

### How can I perform other actions on a SQL Server or Azure SQL Database?

You can use a PowerShell task to execute other types of SQL scripts. For more details, see [Perform SQL server actions in VSTS or TFS](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy your database to Azure SQL database using SQL scripts

2/20/2018 • 4 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

Continuous deployment means starting an automated deployment process whenever a new successful build is available. Here we'll show you how to run SQL scripts against an Azure SQL Database as part of continuous deployment using Release Management.

## Get set up

### Azure SQL Database

Carry out the following steps to set up an Azure SQL Database server against which the database script will be executed.

1. Sign into the Azure management portal and choose the **+New** icon in the left panel, then choose **Data + Storage**. Select **SQL Database** from the list.
2. In the **SQL Database** blade, enter a name for Azure SQL Database and then choose **Server** to configure the required settings for the server.
3. In the **Server** blade, choose **Create a new server**.
4. In the **New server** blade, enter a name for the server and enter the admin login and password for the new server. Leave all other settings as they are and choose **OK**.
5. Back in the **SQL Database** blade, leave all the other settings at their default values and choose **Create**.
6. After the Azure SQL Database server and database have been created, open its blade and make a note of the **Server name**.

### Azure SQL scripts

You will need a **SQL script** that is published as an artifact by Team Build or a similar continuous integration service. Alternatively, you can also store the script in a version control repository. If you do not have a script, follow these steps to get started with a simple script.

- Check the following script into a version control repository such as VSTS or TFS as `DatabaseExample.sql`.

```
USE [master]
GO
IF NOT EXISTS (SELECT name FROM master.sys.databases WHERE name = N'DatabaseExample')
CREATE DATABASE [DatabaseExample]
GO
```

In addition, you will need Azure Powershell scripts to create and remove firewall rules in Azure. Without the firewall rules, the agent cannot communicate with the Azure SQL Database.

- Check the following PowerShell script into the same location that has your SQL script as `SetAzureFirewallRule.ps1`.

```

[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
    [String] [Parameter(Mandatory = $true)] $ServerName,
    [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

function New-AzureSqlServerFirewallRule {
    $agentIP = (New-Object net.webclient).downloadstring("http://checkip.dyndns.com") -replace "[^\d\.]"
    New-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIp -EndIPAddress $agentIp -RuleName
    $AzureFirewallName -ServerName $ServerName
}
function Update-AzureSqlServerFirewallRule{
    $agentIP= (New-Object net.webclient).downloadstring("http://checkip.dyndns.com") -replace "[^\d\.]"
    Set-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIp -EndIPAddress $agentIp -RuleName
    $AzureFirewallName -ServerName $ServerName
}

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -RuleName $AzureFirewallName -
ErrorAction SilentlyContinue) -eq $null)
{
    New-AzureSqlServerFirewallRule
}
else
{
    Update-AzureSqlServerFirewallRule
}

```

- Check the following PowerShell script into the same location that has your SQL script as

`RemoveAzureFirewallRule.ps1`.

```

[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
    [String] [Parameter(Mandatory = $true)] $ServerName,
    [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -RuleName $AzureFirewallName -
ErrorAction SilentlyContinue))
{
    Remove-AzureSqlDatabaseServerFirewallRule -RuleName $AzureFirewallName -ServerName $ServerName
}

```

[How can I perform other actions on a SQL Server or Azure SQL Database?](#)

## Define and test your CD release process

- Open the **Releases** tab of the **Build & Release** hub, open the + drop-down in the list of release definitions, and choose **Create release definition**.
- Select the **Empty** template.
- Select the build definition or repository that contains the SQL scripts as the artifact source.
- Select the **Continuous deployment** check box, and then choose **Create**.
- Add the following tasks to the definition:

## Deploy: Azure Powershell - Add a firewall rule in Azure to allow it to connect to Azure SQL Database.

- **Azure Connection Type:** `Azure Classic`. The scripts provided as samples here work with only an Azure Classic connection.
- **Azure Subscription:** Select an Azure subscription. If you have not created an Azure classic endpoint, create one now by choosing **Add**.
- **Script Type:** `Script File Path`.
- **Script Path:** Select the location of `SetAzureFirewallRule.ps1`.
- **Script Arguments:** Name of the SQL server you created earlier.

## Utility: Command Line - Run the SQL script.

- **Tool:** `SQLCMD`.
- **Arguments:**

```
-S {database-server-name}.database.windows.net -U {username}@{database-server-name} -P {password}
-d {database-name} -i {SQL file}
```

. For example, when the SQL script is coming from an artifact source, **{SQL file}** will be of the form:  
`$(System.DefaultWorkingDirectory)/contoso-repo/DatabaseExample.sql`.

## Deploy: Azure Powershell - Remove the firewall rule in Azure.

- **Azure Connection Type:** `Azure Classic`. The scripts provided as samples here work with only an Azure Classic connection.
- **Azure Subscription:** Select an Azure subscription. If you have not created an Azure classic endpoint, create one now by choosing **Add**.
- **Script Type:** `Script File Path`.
- **Script Path:** Select the location of `RemoveAzureFirewallRule.ps1`.
- **Script Arguments:** Name of the SQL server you created earlier.

6. Edit the name of the release definition, choose **Save**, and choose **OK**. Note that the default environment is named Environment1, which you can edit by clicking directly on the name.

You're now ready to create a release, which means to start the process of running the release definition.

1. Choose **+ Release** and select **Create Release**.
2. Select the build we just completed in the highlighted drop-down list and choose **Create**.
3. Choose the release link in the popup message. For example: "Release **Release-1** has been created".
4. Open the **Logs** tab to watch the release console output.

## Q&A

### How can I perform other actions on a SQL Server or Azure SQL Database?

You can use a PowerShell task to execute other types of SQL scripts. For more details, see [Perform SQL server actions in VSTS or TFS](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Configure and manage VMs with System Center Virtual Machine Manager (SCVMM)

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

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

This example shows how you can integrate **System Center Virtual Machine Manager** (SCVMM) with Release Management in VSTS or Team Foundation Server.

## Prepare

You need SCVMM. If you want to create an isolated virtual network using SCVMM, see [this topic](#).

1. Install the **Virtual Machine Manager** (VMM) console by following [these instructions](#). Supported version: [System Center 2012 R2 Virtual Machine Manager](#).
2. Install an agent on the **agent machine**:
  - [Deploy an agent on Windows](#)
  - [Deploy an agent on macOS](#)
  - [Deploy an agent on Linux](#)
3. Install the **System Center Virtual Machine Manager (SCVMM)** extension from Visual Studio Marketplace into your server or account:
  - If you are using **VSTS**, 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.
4. Create an SCVMM service endpoint in your team project:
  - Open your VSTS or TFS team project in your web browser. Choose the **Settings** icon in the menu bar and select **Services**.
  - In the **Services** tab, choose **New Service Endpoint**, 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 endpoint 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.
5. Choose **OK** to save the settings and create the connection.

## Deploy

1. Open the **Releases** tab of the **Build & Release** hub and choose the "+" icon to create a new release

definition.

2. In the **Create release definition** 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 definition with one default environment and no linked artifacts.
4. Choose **+ Add tasks** and add an **SCVMM** task from the **Deploy** section of the **Task catalog** dialog to the environment.
5. You can select the action from the list of actions available in the task. See [SCVMM task actions](#) for details.
6. You can now add other tasks to the environment, such as **PowerShell on Target Machines** and then deploy to the newly provisioned machines.

In the future, we plan for the **SCVMM** task to provide an output variable that you set in the task and then use as input to subsequent tasks. Until then, if you want to run additional tasks, you must specify the fully-qualified domain names of the virtual machines that are provisioned in SCVMM.

7. Type a name for the new release definition and save it.
8. Create a new release from the release definition and deploy it to the environment.

## See also

- [Create a virtual network isolated environment for build-deploy-test scenarios](#)
- [Task actions for managing VMs using SCVMM](#)

## Q&A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Task actions for managing VMs using System Center Virtual Machine Manager (SCVMM)

2/12/2018 • 5 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

You can manage your virtual machines using the System Center Virtual Machine Manager (**SCVMM**) task by performing a range of actions such as:

- [Create new virtual machines from a template, VHD, or stored VM](#)
- [Delete virtual machines](#)
- [Start and stop virtual machines](#)
- [Create, restore, and delete checkpoints](#)
- [Run custom PowerShell scripts for SCVMM](#)

You must install the **System Center Virtual Machine Manager (SCVMM)** extension from Visual Studio Marketplace into your server or account. For more information, see [Configure and deploy with SCVMM](#).

## Create new virtual machines from a template, VHD, or stored VM

- **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.

SCVMM ⓘ

Version 1.\*

Display name \*

SCVMM : Configure with SCVMM task

SCVMM Service Connection \* ⓘ | [Manage](#) ↗

MySCVMMConnection

Action \*

New Virtual Machine using Template/stored VM/VHD

VM Template options ^

Create virtual machines from VM Templates ⓘ

Stored VM options ^

Create virtual machines from stored VMs ⓘ

VHD options ^

Create virtual machines from vhd ⓘ

Override existing VM Network settings ^

Clear existing network adapters ⓘ

Create VM options ^

Deploy the VMs to ⓘ

Cloud

Cloud Name \*

MyCloud

Additional Arguments ⓘ

`-StartVM -StartAction NeverAutoTurnOnVM -StopAction SaveVM`

Wait Time ⓘ

600

Network Virtualization options ^

## Delete virtual machines

- **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.

The screenshot shows the 'SCVMM' configuration dialog box. It includes the following fields:
 

- Display name \***: SCVMM : Configure with SCVMM task
- SCVMM Service Connection \***: MySCVMMConnection
- Action \***: Delete Virtual Machine
- VM Names \***: VM1
- Select VMs From**: None
- Control Options**: (dropdown menu)

## Start and stop virtual machines

- **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.

SCVMM ⓘ

Version 1.\*

Display name \*

SCVMM : Configure with SCVMM task

SCVMM Service Connection \* ⓘ | Manage ↗

MySCVMMConnection

Action \*

Start Virtual Machine

VM Names \*

VM1

Select VMs From ⓘ

None

Wait Time ⓘ

600

Control Options

This screenshot shows the configuration page for a SCVMM task. It includes fields for the task's display name, the SCVMM service connection it uses, the specific action (like starting a virtual machine), the names of the virtual machines involved, and a wait time for the operation to complete.

## Create, restore, and delete checkpoints

- **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.

SCVMM ⓘ

Version 1.\*

Display name \*

SCVMM : Configure with SCVMM task

SCVMM Service Connection \* ⓘ | Manage ↗

MySCVMMConnection

Action \*

Create Checkpoint

VM Names \*

VM1

Checkpoint Name \*

CP1

Description For Checkpoint ⓘ

First checkpoint

Select VMs From ⓘ

None

Control Options ▾

## Run custom PowerShell scripts for SCVMM

- **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.

SCVMM ⓘ

Version 1.\* ▾

Display name \*

SCVMM : Configure with SCVMM task

SCVMM Service Connection \* ⓘ | Manage ↗

MySCVMMConnection ▾ ⚡ + New

Action \* ⓘ

Run Powershell Script For SCVMM

Script Type \* ⓘ

Script File Path

Script Path \* ⓘ

`$(System.DefaultWorkingDirectory)/SampleWebAppWithTestsBuild/drop/myscript.ps1`

Script Arguments ⓘ

`-appname MyNewApp -apptype SalesOffice`

Working folder ⓘ

C:/temp

Control Options ▾

This screenshot shows the configuration interface for a 'Configure with SCVMM task'. The 'Action' dropdown is set to 'Run Powershell Script For SCVMM'. The 'Script Path' field contains the PowerShell script path: '\$(System.DefaultWorkingDirectory)/SampleWebAppWithTestsBuild/drop/myscript.ps1'. The 'Script Arguments' field contains '-appname MyNewApp -apptype SalesOffice'. The 'Working folder' is set to 'C:/temp'. There are also sections for 'Control Options' and 'Script Type'.

## See also

- [Configure and deploy with SCVMM](#)
- [Create a virtual network isolated environment for build-deploy-test scenarios](#)

## Q&A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Provision and manage virtual machines in VMware

1/19/2018 • 6 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

This example shows how you can integrate **VMware vCenter Server** with Release Management in Visual Studio Team Services (VSTS) or Team Foundation Server (TFS). You can provision and manage virtual machines in vCenter and then deploy your apps to them.

## Prepare

You will need a minimum of two machines - a **target server** or virtual machine to deploy to, and an **agent machine** that drives the deployment. You may have multiple target servers depending on how many nodes you want to deploy to. However, you only need one agent machine to drive the deployment.

1. The extension uses 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`

1. Install an agent on the **agent machine**:

- [Deploy an agent on Windows](#)
- [Deploy an agent on macOS](#)
- [Deploy an agent on Linux](#)

2. Install the VMware extension from Visual Studio Marketplace into your server or account.

- If you are using **VSTS**, install the extension from [this location](#) in Visual Studio Marketplace.
- If you are using **Team Foundation Server**, download the extension from [this location](#) in Visual Studio Marketplace, upload it to your Team Foundation Server, and install it.

3. Follow these steps to create a vCenter Server Service endpoint in your team project:

- Open your VSTS or TFS team project in your web browser. Choose the **Settings** icon in the menu bar and select **Services**.
- In the **Services** tab, choose **New Service Endpoint**, 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 endpoint 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.

4. Choose **OK** to save the settings and create the connection.

## Deploy

1. Open the **Releases** tab of the **Build & Release** hub and choose the "+" icon to create a new release definition.
2. In the **Create release definition** 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 definition with one default environment and no linked artifacts.
4. Choose **+ Add tasks** and add a **VMware Resource Deployment** task from the **Deploy** section of the **Task catalog** dialog to the environment.
5. To configure the **VMware Resource Deployment** task to take snapshot of virtual machines, or to revert or delete them, use these settings:



**VMWare Resource Deployment** - Connect to a VMware vCenter Server, easily provision VMs, and perform actions on 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).

6. To configure the **VMware Resource Deployment** task to provision a new virtual machine from a template, use these settings:



**VMWare Resource Deployment** - Connect to a VMware vCenter Server, easily provision VMs, and perform actions on them.

- **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.

7. You can now add other tasks to the environment, such as **PowerShell on Target Machines**, and then deploy to the newly provisioned machines.

In the future we plan for the **VMware Resource Deployment** task to provide an output variable that you set in the task and then use as input to subsequent tasks. Until then, if you want to run additional tasks, you'll need to specify the fully-qualified domain names of the virtual machines that are provisioned in VMware.

8. Type a name for the new release definition and save it.
9. Create a new release from the release definition and deploy it to the environment.

## Q&A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.

- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Continuous Testing

2/26/2018 • 1 min to read • [Edit Online](#)

[VSTS | TFS 2018 | TFS 2017 | TFS 2015](#) | [Previous version](#)

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.

## 5-Minute Quickstarts

		
Test with builds	Review test results	

## Videos

<a href="https://channel9.msdn.com/Events/Connect/2017/T182/player">https://channel9.msdn.com/Events/Connect/2017/T182/player</a>	<a href="https://channel9.msdn.com/Events/Build/2016/P581/player">https://channel9.msdn.com/Events/Build/2016/P581/player</a>
<a href="https://channel9.msdn.com/Events/Seth-on-the-Road/That-Conference-2016/Creating-a-Software-Immune-System-for-Testing-under-DevOps/player">https://channel9.msdn.com/Events/Seth-on-the-Road/That-Conference-2016/Creating-a-Software-Immune-System-for-Testing-under-DevOps/player</a>	<a href="https://channel9.msdn.com/Series/Parts-Unlimited-Labs/Parts-Unlimited-Continuous-Integration/player">https://channel9.msdn.com/Series/Parts-Unlimited-Labs/Parts-Unlimited-Continuous-Integration/player</a>
<a href="https://channel9.msdn.com/Events/TechDaysOnline/UK-TechDays-Online-September-2016/Supporting-DevOps-through-testing--its-not-just-about-automating-tests/player">https://channel9.msdn.com/Events/TechDaysOnline/UK-TechDays-Online-September-2016/Supporting-DevOps-through-testing--its-not-just-about-automating-tests/player</a>	

## Step-by-Step Tutorials

- [Run unit tests and load tests as you build and deploy your app](#)

## How-to Guides

- [Set up continuous testing](#)
- [Run tests from the Test hub](#)
- [Associate tests with test cases](#)
- [Associate results with requirements](#)
- [Speed up testing with Test Impact Analysis](#)
- [Run tests in parallel](#)

## Reference

- [Unified agents and phases](#)
- [Test Java applications](#)
- [Set up environments](#)

- [FAQs](#)

## Resources

- [DevOps Rangers](#)
- [Plug-ins for Visual Studio](#)
- [Manual and exploratory testing](#)
- [Load and performance testing](#)
- [Unit testing](#)

# Publish symbols for debugging

2/14/2018 • 1 min to read • [Edit Online](#)

## NOTE

A symbol server is available with Package Management in **VSTS** and works best with **Visual Studio 2017 Update 4 or later**.

**Team Foundation Server** users and users without the Package Management extension can publish symbols to a file share using a [build task](#).

## IMPORTANT

To use symbol server, ensure you've [installed Package Management](#) and assigned a license for each user that will consume symbols.

Symbol servers enable debuggers to automatically retrieve the correct symbol files without knowing product names, build numbers or package names. To learn more about symbols, read the [concept page](#); to consume symbols, see [this page for Visual Studio](#) or [this page for WinDbg](#).

## Publish symbols

In order to publish symbols to the Package Management symbol server in VSTS, include the [Index Sources and Publish Symbols](#) task in your build definition. Configure the task as follows:

- For **Version**, select the 2.\* (preview).
- For **Symbol Server Type**, select **VSTS**.
- 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`.

Index Sources & Publish Symbols (Preview) ⓘ

Version 2.\* (preview) ⚙️ Link settings X Remove

Display name \*

Publish symbols path

Path to symbols folder ⓘ

\$(Build.SourcesDirectory)

Search pattern \* ⓘ

\*\*/bin/\*\*/\*.pdb

Index sources ⓘ

Publish symbols ⓘ

Symbol Server Type \* ⓘ

Team Services

Advanced

Control Options

## Publish symbols for NuGet packages

To publish symbols for NuGet packages, include the above task in the build definition that produces the NuGet packages. Then the symbols will be available to all users in the VSTS account.

## Q&A

**Q: What's the retention policy for the symbols stored in the VSTS symbol server?**

A: Symbols will have the same retention as the build. When you delete a build, you also delete the symbols produced by that build.

**Q: Can I use source indexing on a portable PDB created from a .NET Core assembly?**

A: No, source indexing is currently not enabled for Portable PDBs as SourceLink doesn't support authenticated source repositories. The workaround at the moment is to configure the build to generate full PDBs.

**Q: Is this available in TFS?**

A: In TFS, you can bring your own file share and set it up as a symbol server as described in [this blog](#).

# Restore Package Management NuGet packages in Team Build

1/9/2018 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017

This walkthrough will cover setting up an existing build to restore NuGet packages from Package Management feeds. It assumes that you've already:

- [Set up your solution](#) to consume packages from a Package Management feed
- [Created a build for that solution](#)
- [Added the correct build service identity](#) to your feed

To build a solution that relies on NuGet packages from Package Management feeds, add the **NuGet** task (if one is not already present).

First, click **Add build step...**, select the **Package** category, and add the **NuGet** task. Then drag to order the task above any build tasks that require your packages.

Next, configure these options:

- **Command:** restore
- **Path to solution, packages.config, or project.json:** The path to the file that specifies the packages you want to restore

Then, select feeds to use:

- If you've checked in a [NuGet.config](#), select **Feeds in my NuGet.config** and select the file from your repo.
- If you're using a single VSTS/TFS feed, select the **Feed(s) I select here** option and select your feed from the dropdown.

The screenshot shows the VSTS/TFS build process editor. On the left, the 'Process' pane lists build steps: 'Get sources', 'NuGet restore' (selected), 'Build solution \*\*\\*.sln', 'VsTest - testAssemblies', 'Publish symbols path:', 'Copy Files to: \$(build.a...)', 'Publish Artifact: drop', and '+ Add Task'. On the right, the 'NuGet' task configuration pane is open, showing the following settings:

- Display name:** NuGet restore
- Command:** restore
- Path to solution, packages.config, or project.json:** \*\*\\*.sln
- Feeds and authentication:** Feeds to use:
  - Feed(s) I select here
  - Feeds in my NuGet.config
- Use packages from this VSTS/TFS feed:** FabrikamFiber
- Use packages from NuGet.org:**

Finally, save your build.

# Specifying sources in NuGet.config

The NuGet.config you check in should list all the package sources you want to consume. The example below demonstrates how that might look.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <!-- remove any machine-wide sources with <clear/> -->
    <clear />
    <!-- add a VSTS feed -->
    <add key="MyGreatFeed"
      value="https://fabrikam.pkgs.visualstudio.com/DefaultCollection/_packaging/MyGreatFeed/nuget/v3/index.json" />
    <!-- also get packages from the NuGet Gallery -->
    <add key="nuget.org" value="https://www.nuget.org/api/v2/" />
  </packageSources>
  <activePackageSource>
    <add key="All" value="(Aggregate source)" />
  </activePackageSource>
</configuration>
```

## Q & A

### Why can't my build restore packages?

NuGet restore can fail due to a variety of issues. One of the most common issues is the introduction of a new project in your solution that requires a [target framework](#) that isn't understood by the version of NuGet your build is using. This issue generally doesn't present itself on a developer machine because Visual Studio updates the NuGet restore mechanism at the same time it adds new project types. We're looking into similar features for VSTS. 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 VSTS or the upcoming TFS 2018 release, 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 step to select the version of NuGet for all the subsequent steps. You can see all available versions of NuGet [on nuget.org](#).

The screenshot shows the TFS Build and Release interface. The top navigation bar includes 'FabrikamFiber', 'Build and Release', 'Search work items', and user profile icons. Below the navigation is a secondary menu with links to 'Builds', 'Releases', 'Packages', 'Library', 'Task Groups', and 'Deployment Groups\*'. The main content area displays a build pipeline named 'FabrikamFiber-Cl'. The 'Tasks' tab is active, showing the following steps:

- Get sources
- Use NuGet 4.3.0 (selected)
- NuGet restore

The 'Use NuGet 4.3.0' step is currently being configured. Its properties pane shows:

- Version: 0.\*
- Display name: Use NuGet 4.3.0
- Version of NuGet.exe to install: 4.3.0
- Always download the latest matching version:

Buttons for 'Link settings' and 'Remove' are also 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 Team Build.

1. Add the task, if you haven't already. If you have a "NuGet Restore" step in the catalog (it may be in the Deprecated tasks section), insert it into your build. Otherwise, insert a "NuGet" step.
2. For your NuGet/NuGet Installer step, 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 step, add a "PowerShell" step, 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.

# Publish NuGet packages from Team Build to Package Management

12/19/2017 • 2 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017

This walkthrough will cover packing and publishing .NET Framework NuGet packages from Package Management feeds. It assumes that you've already:

- [Created a build](#) for a .NET Framework solution that produces a set of DLLs and/or other content you wish to package and publish
- [Added the correct build service identity](#) to your feed

## Creating a NuGet package

There are a variety of ways to create NuGet packages during a build. If you're already using MSBuild or some other step to create your packages, skip this section and [publish those packages](#). Otherwise, add a **NuGet** task.

First, click **Add build step...**, select the **Package** category, and add the **NuGet** task. Then drag to order the task below your Visual Studio Build (or similar) task and above any build tasks that require the packages you build.

Next, configure these options:

- **Command:** pack
- **Path to csproj or nuspec file(s) to pack:** 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 to package:** Leave this as \$(BuildConfiguration) unless you wish to always build either Debug or Release packages, or unless you have a custom build configuration.
- **Package folder:** Leave this as \$(Build.ArtifactStagingDirectory). If you change this, make a note of the location so you can use it in the [publish step](#).
- **Pack options > Use Build number to version package:** See the [next section](#)

The screenshot shows the Azure DevOps build pipeline configuration. On the left, a sidebar lists various build steps: Get sources, NuGet restore, Build solution, NuGet pack (which is selected and highlighted in blue), VsTest - testAssemblies, Publish symbols path, Copy Files to: \$(build.artifactstagingdirectory), Publish Artifact: drop, and Add Task. On the right, the 'NuGet' task configuration page is displayed. It includes fields for Display name (set to 'NuGet pack'), Command (set to 'pack'), Path to csproj or nuspec file(s) to pack (set to '\*\*/\*.csproj'), Configuration to package (set to '\$(BuildConfiguration)'), Package folder (set to '\$(Build.ArtifactStagingDirectory)'), and Pack options (with Automatic package versioning set to 'Off').

Finally, save your build.

### Use the build number to version package

Checking the **Use Build number to version package** box requires a change to your build's versioning scheme. Work is planned to improve this scenario; right now, it's best to version packages via other mechanisms (like using the `AssemblyVersionAttribute` from the `csproj`, as outlined in the [NuGet docs](#)).

If you choose to use the build number to version your package, you'll need to increment the package version for continuous integration builds. This is because specific versions of packages in feeds are [immutable](#) and so cannot be updated or replaced.

To use the build number, check the **Use build number to version package** box and follow the line's instructions (hover over the blue *i* icon) to set the build version format string. You must set the build version format string to have at least three parts separated by periods to avoid an error in NuGet packaging. The default build version format string is `$(date:yyyyMMdd)$(rev:.r)`, so a simple change is to add a zero at the end and a period between the date and build counter: `$(date:yyyyMMdd).$(rev:.r).0`.

Don't forget to save your build.

## Publish packages created by your build

To publish NuGet packages created by your build to Package Management feeds, add the **NuGet** task. This section assumes that a previous task (for example, a **NuGet** task set to the **pack** command) in your build is already producing NuGet packages.

First, click **Add build step...**, select the **Package** category, and add the **NuGet** task. Then drag to order the task below the build task producing your NuGet packages.

Next, configure these options:

- **Command:** push

- **Path to NuGet package(s) to publish:** Leave this as \$(Build.ArtifactStagingDirectory) unless you elected earlier to pack your packages in another location in the last step.
- **Target feed location:** This account/collection
- **Target feed:** Select the feed you want to publish to

The screenshot shows the Azure DevOps build pipeline editor. On the left, a sidebar lists various build steps: Get sources, NuGet restore, Build solution, NuGet pack, NuGet push (which is selected and highlighted in blue), VsTest - testAssemblies, Publish symbols path, Copy Files to, and Publish Artifact: drop. On the right, the configuration details for the selected 'NuGet push' step are displayed. The step is version 2.\* and has a display name of 'NuGet push'. The command is set to 'push'. The path to NuGet package(s) to publish is '\$(Build.ArtifactStagingDirectory)/\*.nupkg'. The target feed location is set to 'This account/collection' (FabrikamFiber). The target feed is 'FabrikamFiber'. There is also an unchecked checkbox for 'Allow duplicates to be skipped'.

Finally, save your build.

## Publish symbols for your packages

When you push packages to a Package Management feed, you can also [publish symbols](#).

# Use Jenkins to restore and publish packages

12/19/2017 • 3 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017

VSTS's package management works with the continuous integration tools your team already uses. In this [Jenkins](#) walkthrough, you'll create a NuGet package and publish it to a VSTS 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:

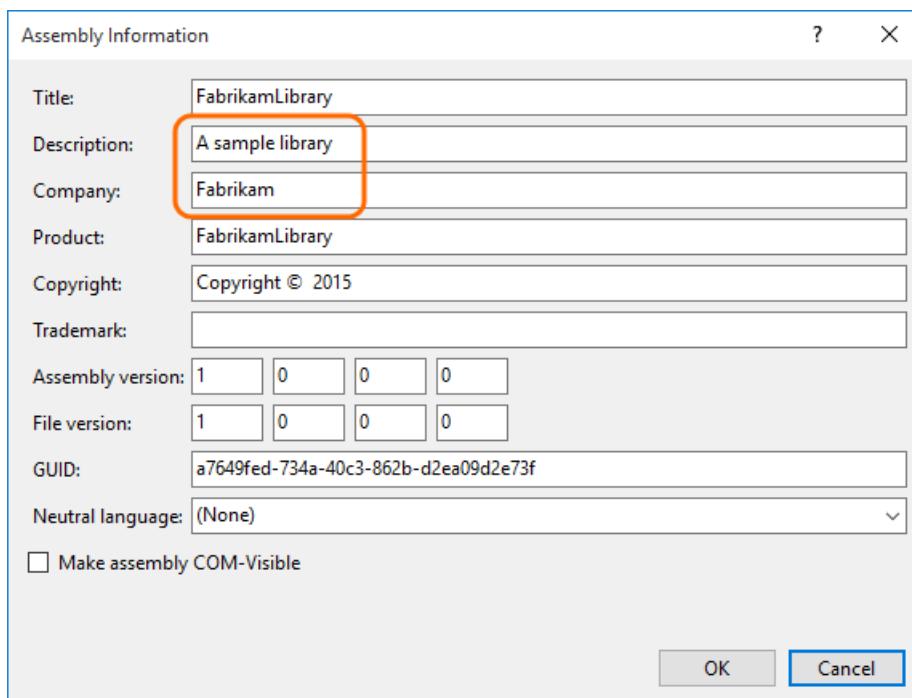
- [MSBuild 1.24](#)
- [Git 2.4.0](#)
- [Git Client 1.19.0](#)
- [Credentials Binding plugin 1.6](#)

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 VSTS NuGet tools to your repo

The easiest way to use the VSTS 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 VSTS 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 VSTS and add it to your project

- [Create a feed](#) in your VSTS account called *MyGreatFeed*. Since you're the owner of the feed, you will automatically be able to push packages to it.
- Add the URL for the feed you just generated to the `nuget.config` in the root of your repo.
  - Find the `<packageSources>` section of `nuget.config`.
  - Just before `</packageSources>`, add a line using this template:  
`<add key="MyGreatFeed" value="{feed_url}" />`. Change `{feed_url}` to the URL of your feed.
  - Commit this change to your repo.

```

1  <?xml version="1.0" encoding="utf-8"?>
2  <configuration>
3    <packageSources>
4      <clear />
5      <add key="vss-package-management" value=
6        "https://www.myget.org/F/vss-package-management/api/v2" />
7      <add key="MyGreatFeed" value=
8        "https://fabrikam.pkgs.visualstudio.com/DefaultCollection/_apis/packaging/MyGr
9        eatFeed/nuget/index.json" />
10     </packageSources>
11     <activePackageSource>
12       <add key="All" value="(Aggregate source)" />
13     </activePackageSource>
14   </configuration>

```

- [Generate a PAT \(personal access token\)](#) for your user account. This PAT will allow the Jenkins build process to authenticate to VSTS 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 definition in Jenkins

- Ensure you have the [correct plugins installed in Jenkins](#).
- This will be a Freestyle project. Call it "Fabrikam.Walkthrough".

 Jenkins

Jenkins > All >

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**

 [Help us localize this page](#) Page generated: Oct 27, 2015 8:42:49 AM [REST API](#) [Jenkins ver. 1.635](#)

- 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

Scope

Username

Password

Description

## Build Environment

Use secret text(s) or file(s)



### Bindings

#### Username and password (separated)



Username Variable



Password Variable



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 step 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 definition 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 definition in Jenkins.
- After the last build step (which runs `nuget pack`), add a new **Execute a Windows batch command** build step.
- 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 VSTS and push the package to the feed you selected.

# Use Team Build to restore and publish npm packages

9/26/2017 • 1 min to read • [Edit Online](#)

## VSTS

This guide covers the basics of using Team Build to work with npm packages in Package Management feeds.

This walkthrough assumes that you've already:

- [Set up your `npmrc` files](#) to point to a Package Management feed
- Created a build
- [Added the correct build service identity](#) to your feed

## Install packages at the start of your build

To build a solution that relies on npm packages from Package Management feeds, add the **npm** task.

First, click **Add build step...**, select the **Package** category, and add the **npm** task. Then drag to order the task above any build tasks that require your packages.

Next, configure these options:

- **working folder:** Select the folder that contains your `.npmrc`; leave blank if your `.npmrc` is at the root of the repo
- **npm command:** `install`



Finally, save your build.

## Publish a package

To publish an npm package to a Package Management feed, add the **npm** task.

First, click **Add build step...**, select the **Package** category, and add the **npm** task. Then drag to order the task above any build tasks that require your packages.

Next, configure these options:

- **working folder:** Select the folder that contains your `.npmrc` and `package.json`; leave blank if those files are at the root of the repo
- **npm command:** `publish`

## npm publish

working folder	<input type="text"/>	 
npm command	<input type="text" value="publish"/>	
arguments	<input type="text"/>	

Finally, save your build.

### NOTE

Team Build does not support using the `publishConfig` property to specify the `registry` to which you're publishing. Ensure your working folder has a `.npmrc` file with a `registry=` line, as detailed in the Connect to feed screen in your feed.

# Set up Team Build and Maven

1/31/2018 • 1 min to read • [Edit Online](#)

This guide covers the basics of using Team Build to work with Maven artifacts in Package Management feeds.

This walkthrough assumes that you've already added the correct build service identity to your feed.

1. Create a new build definition and select the **Maven** template.
2. Fill in the path to your `POM.xml`, configure the Maven goal you'd like for your build. Authentication to your VSTS feed should happen automatically.

# Migrate from XAML builds to new builds

2/28/2018 • 12 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | XAML builds

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 VSTS, 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 definition into a new build definition. The migration process is to manually create the new build definitions 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 definition.
3. Create a new build definition intended to replace one of your XAML build definitions.
  - a. Create a new build definition.
  - b. Port your XAML settings.
4. On the [General tab](#), disable the XAML build definition.
5. Repeat the previous two steps for each of your XAML build definitions.
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 definitions.

### WARNING

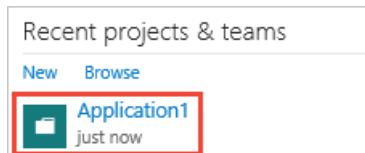
After you delete the XAML builds and definitions, you cannot get them back.

## Create new build definitions

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 definitions in the new build system.

## 1. Open your team project in your web browser □

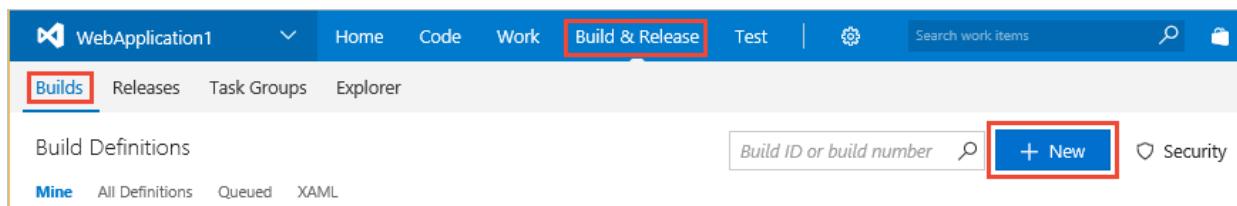


(If you don't see your team project listed on the home page, select **Browse**.)

- On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_team_project}`
- VSTS: `https://{your_account}.visualstudio.com/DefaultCollection/{your_team_project}`

The TFS URL doesn't work for me. How can I get the correct URL?

## 2. Create a build definition (Build and Release tab > Builds) □



3. Select a template to add commonly used tasks to your build definition.
4. Make any necessary changes to your build definition to replicate your XAML build definition. 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 definitions, 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 definition.

### General tab

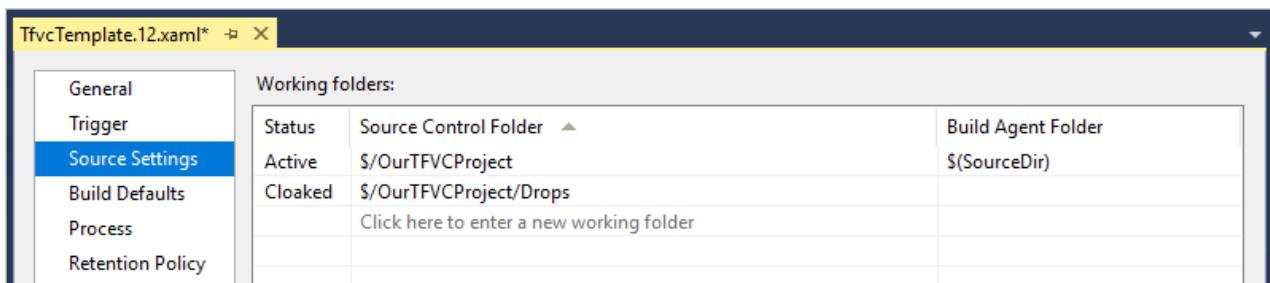
A screenshot of the 'General' tab in the TFS build definition editor. The tab title is 'OurBuild'. The left sidebar lists 'General', 'Trigger', 'Source Settings', 'Build Defaults', 'Process', and 'Retention Policy'. The main area contains fields for 'Build definition name' (set to 'OurBuild') and 'Description (optional)'. Below that is a section for 'Queue processing' with three radio button options: 'Enabled' (selected), 'Paused', and 'Disabled'. Descriptions for each option are provided: 'Enabled' says requests will be added to the queue and start in priority order; 'Paused' says requests will be added but won't start unless forced; 'Disabled' says no requests will be queued or started and it won't participate in triggered builds like Continuous Integration or Gated.

XAML SETTING	TFS 2017 EQUIVALENT	VSTS AND TFS 2018 AND NEWER EQUIVALENT
--------------	---------------------	--

XAML SETTING	TFS 2017 EQUIVALENT	VSTS AND TFS 2018 AND NEWER EQUIVALENT
Build definition name	You can change it whenever you save the definition.	When editing the definition: On the <b>Tasks</b> tab, in left pane click <b>Process</b> , and the <b>Name</b> field appears in right pane.  In the <b>Builds</b> hub ( <b>Mine</b> or <b>All Definitions</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

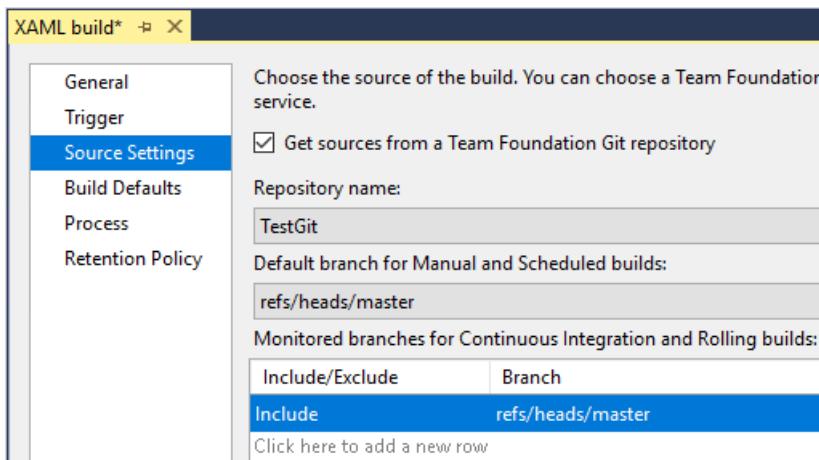
TFVC



XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	VSTS 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 definition offers you some new options. The specific extra options you'll see depend on the version you're using of TFS or VSTS. If you're using VSTS, first make sure to display **Advanced settings**. See [Build definition source repositories](#).

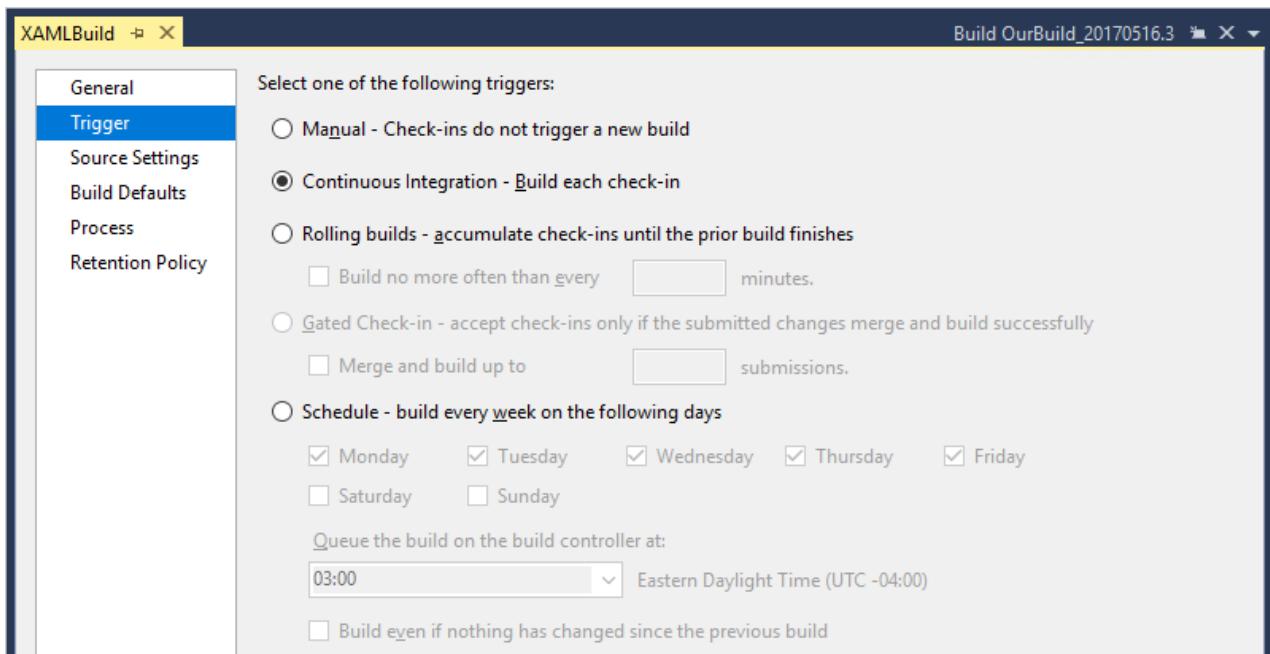
Git



XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	VSTS 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 definition offers you some new options. The specific extra options you'll see depend on the version you're using of TFS or VSTS. If you're using VSTS, first make sure to display **Advanced settings**. See [Build definition source repositories](#).

### Trigger tab



XAML SETTING	TFS 2017 AND NEWER, VSTS EQUIVALENT
Trigger tab	On the <b>Triggers</b> tab, select the trigger you want to use: CI, scheduled, or gated.

The new build definition offers you some new options. For example:

- You can potentially create fewer build definitions to replace a larger number of XAML build definitions. This is because you can use a single new build definition with multiple triggers. And if you're using VSTS, 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 VSTS, you can specify the maximum number of concurrent builds 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 VSTS. See [Build definition triggers](#)

We don't yet support the **Build even if nothing has changed since the previous build** option.

## Build Defaults tab

The screenshot shows the 'XAMLBuild' configuration window with the 'Build Defaults' tab selected. The 'Build controller' dropdown is set to 'Private controller'. The 'Description' field is empty. Under 'Staging location', the radio button 'Copy build output to the server' is selected.

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Build controller	On the <b>General</b> tab, select the default agent queue.	On the <b>Options</b> tab, select the default agent queue.
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 definition 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

The screenshot shows the 'TF Version Control' configuration window with the 'Build process parameters' section expanded. It lists steps 1 through 5 under '1. TF Version Control': 'Clean workspace' (True) and 'Get version' (True). Below these are sections for '2. Build', '3. Test', '4. Publish Symbols', and '5. Advanced'.

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS 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.)

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Get version	You can't specify a changeset in the build definition, but you can specify one when you manually queue a build.	You can't specify a changeset in the build definition, 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 definition offers you some new options. See [Build definition source 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	VSTS 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 definition, but you can specify one when you manually queue a build.	You can't specify a commit in the build definition, but you can specify one when you manually queue a build.

The new build definition offers you some new options. See [Build definition source 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 (VSTS), 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, VSTS 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 Team Build</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 process 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 process</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 definition offers you some new options. See [Visual Studio Build](#).

Learn more: [Visual Studio Build task](#) (for building solutions), [MSBuild task](#) (for building individual projects).

#### Test

Build process parameters:		
> 1. TF Version Control		
> 2. Build		
▽ 3. Test		
▽ 1. Automated tests	1 set(s) of tests specified.	
▽ 1. Test source	- Run tests in test sources matching **\*test*.dll;**\*test*.appx, Target platform X86	
Fail build on test failure	False	
▽ Run settings	Run settings	
Run settings file		
Type of run settings	Default	
Target platform for test execution	X86	
Test case filter		
Test run name		
Test sources spec	**\*test*.dll;**\*test*.appx	
▽ 2. Advanced		
Analyze test impact	True	
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 [Get started with continuous testing](#) and [Visual Studio Test task](#).

#### Publish Symbols

Build process parameters:		
> 1. TF Version Control		
> 2. Build		
> 3. Test		
▽ 4. Publish Symbols		
Path to publish symbols		
> 5. Advanced		

XAML PROCESS PARAMETER	TFS 2017 AND NEWER, VSTS 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	04:00:00	
Name filter	*	
Tag comparison operator	MatchExactly	
Tags filter		
Build number format	\$(BuildDefinitionName)_\$(Date:yyyyMMdd)\$(Rev:.r)	
Create work item on failure	True	
Update work items with build number	True	

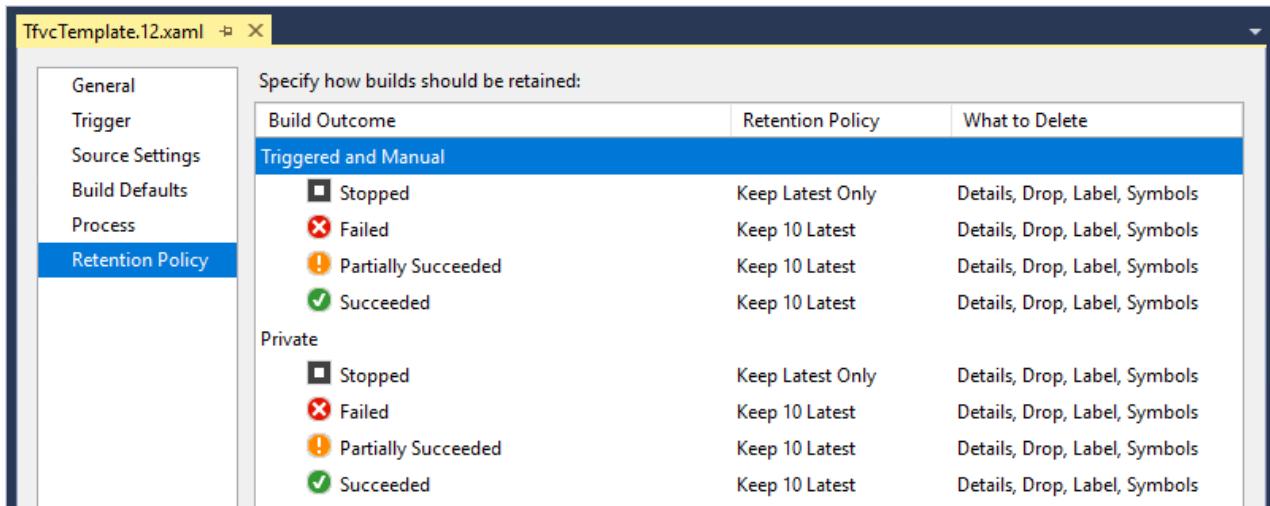
XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Maximum agent execution time	None	On the <b>Options</b> tab you can specify <b>Build job timeout in minutes</b> .

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Maximum agent reservation wait time	None	None
Name filter, Tag comparison operator, Tags filter	A build process asserts demands that are matched with agent capabilities. See <a href="#">Agent capabilities</a> .	A build process 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 definition offers you some new options. See:

- [Agent capabilities](#)
- [Build number format](#)

### Retention Policy tab



XAML PROCESS PARAMETER	TFS 2017 AND NEWER, VSTS EQUIVALENT
Retention Policy tab	On the <b>Retention</b> tab specify the policies you want to implement.

The new build definition 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 VSTS 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 Team Build is tightly integrated with Release Management. So it's easier than ever to automatically kick off a deployment after a successful build. Learn more:

- [CI/CD Hello world](#)
- [Release definitions](#)
- [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 process. 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 definition.

The screenshot shows the TFS build interface. On the left, there's a sidebar with a green plus icon and the text "Add build step...". Below it are two task cards: "Build solution \$/TestTFVC/C Visual Studio Build" and "PowerShell Script". The "PowerShell Script" card is selected and expanded, showing its configuration details. The "Type" field is set to "Inline Script", and the "Script" field contains the PowerShell command "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 definition and [add the tasks you need](#). After you've got a pattern that you like, you can clone it or save it as a template directly in your web browser. See [CI/CD Hello world](#).

### Task groups (TFS 2017 or newer)

In XAML builds, if you change the template, then you also change the behavior of all definitions 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 definition, subsequent changes to the template have no effect on build definitions.

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 definitions that use it to automatically be changed.

## Q&A

### How do I add conditional logic to my build process?

Although the new build definitions 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 VSTS 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/'))
```

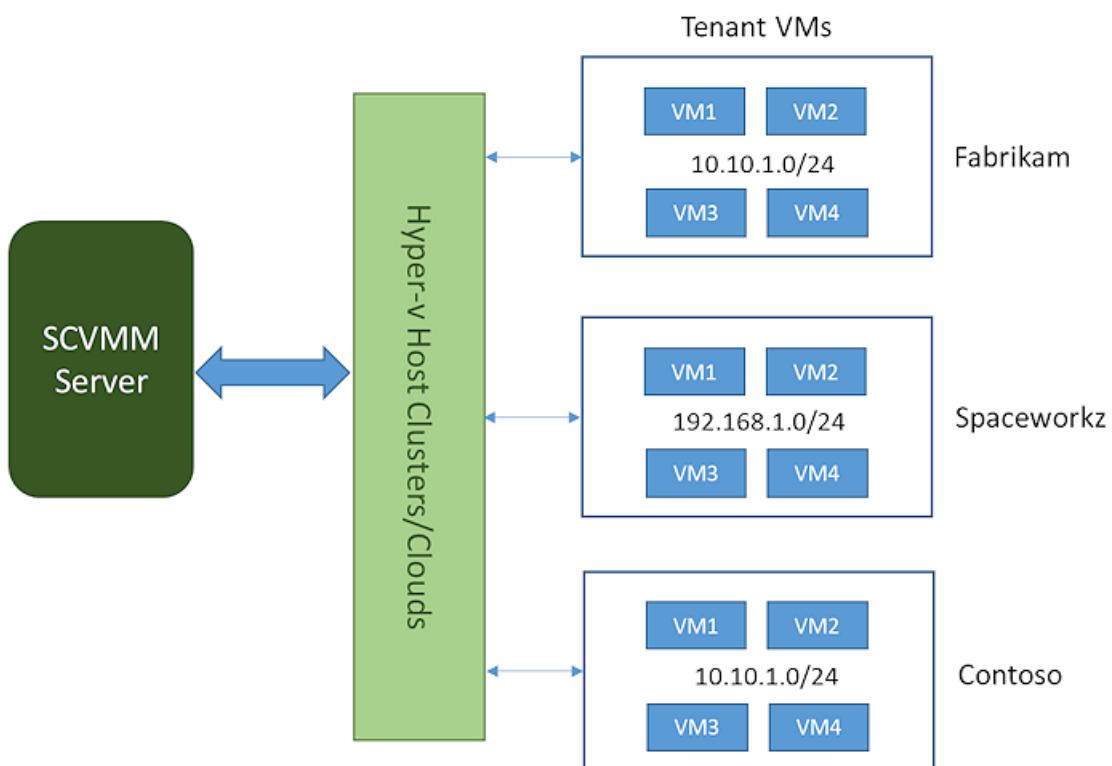
See [Specify conditions for running a task](#).

# Create a virtual network isolated environment for build-deploy-test scenarios

2/12/2018 • 9 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

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](#).

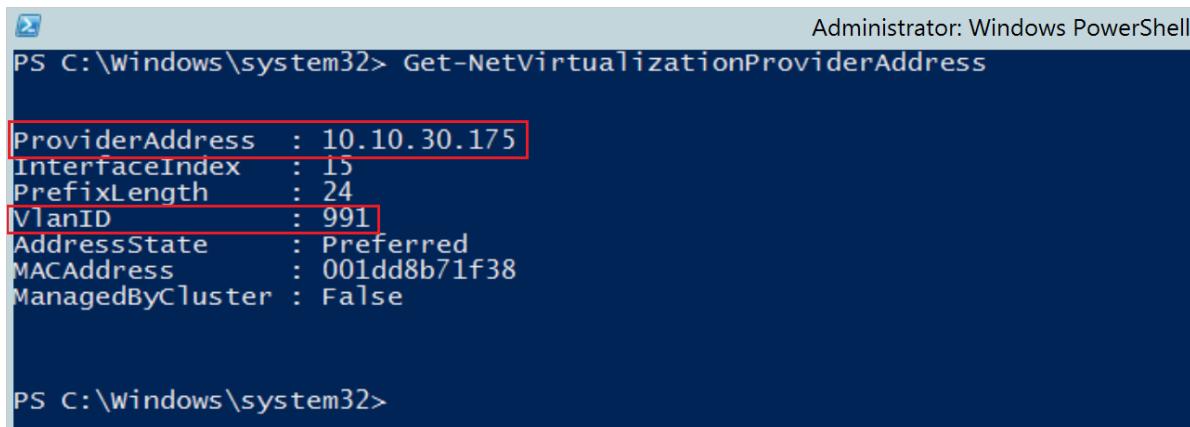
- You can perform a range of operations using SCVMM, see [this topic](#).
- For another example of using SCVMM to manage VMs, see [this topic](#).

## Prerequisites

- SCVMM Server 2012 R2 or later.
- Window 2012 R2 host machines with Hyper-V set up with at least two physical NICs attached.
- One NIC (perhaps external) with corporate network or Internet access.
- One NIC configured in Trunk Mode with a VLAN ID (such as 991) and routable IP subnets (such as 10.10.30.1/24). You network administrator can configure this.
- All Hyper-V hosts in the host group have the same VLAN ID. This host group will be used for your isolated networks.

Verify the setup is working correctly by following these steps:

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.

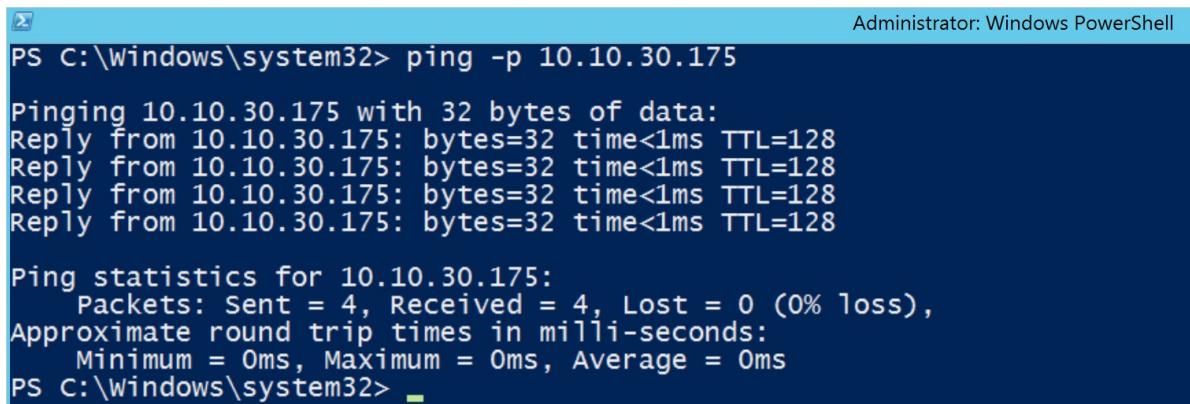


```
Administrator: Windows PowerShell
PS C:\Windows\system32> Get-NetVirtualizationProviderAddress

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](#)

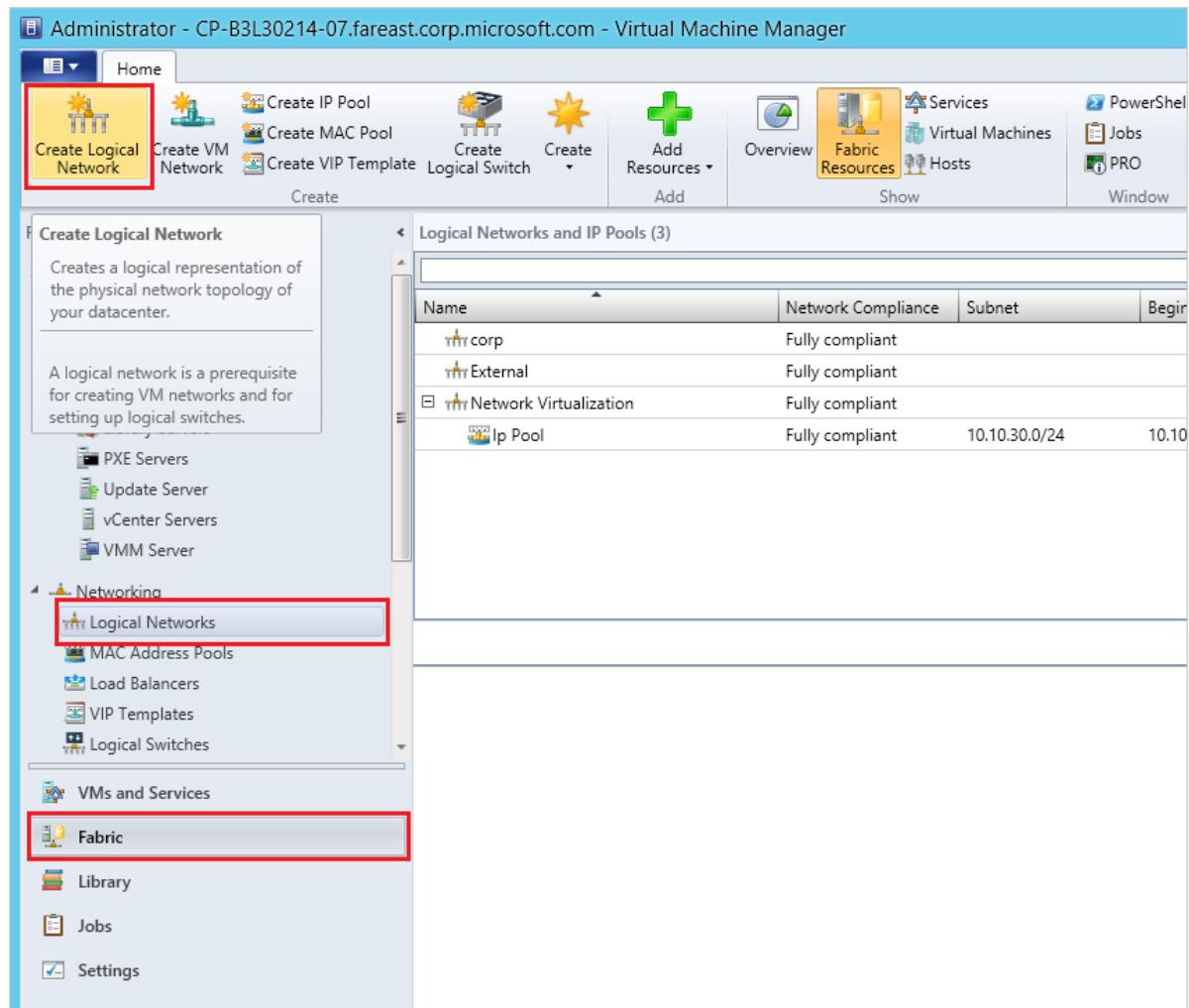
## Create a Network Virtualization layer in SCVMM

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.

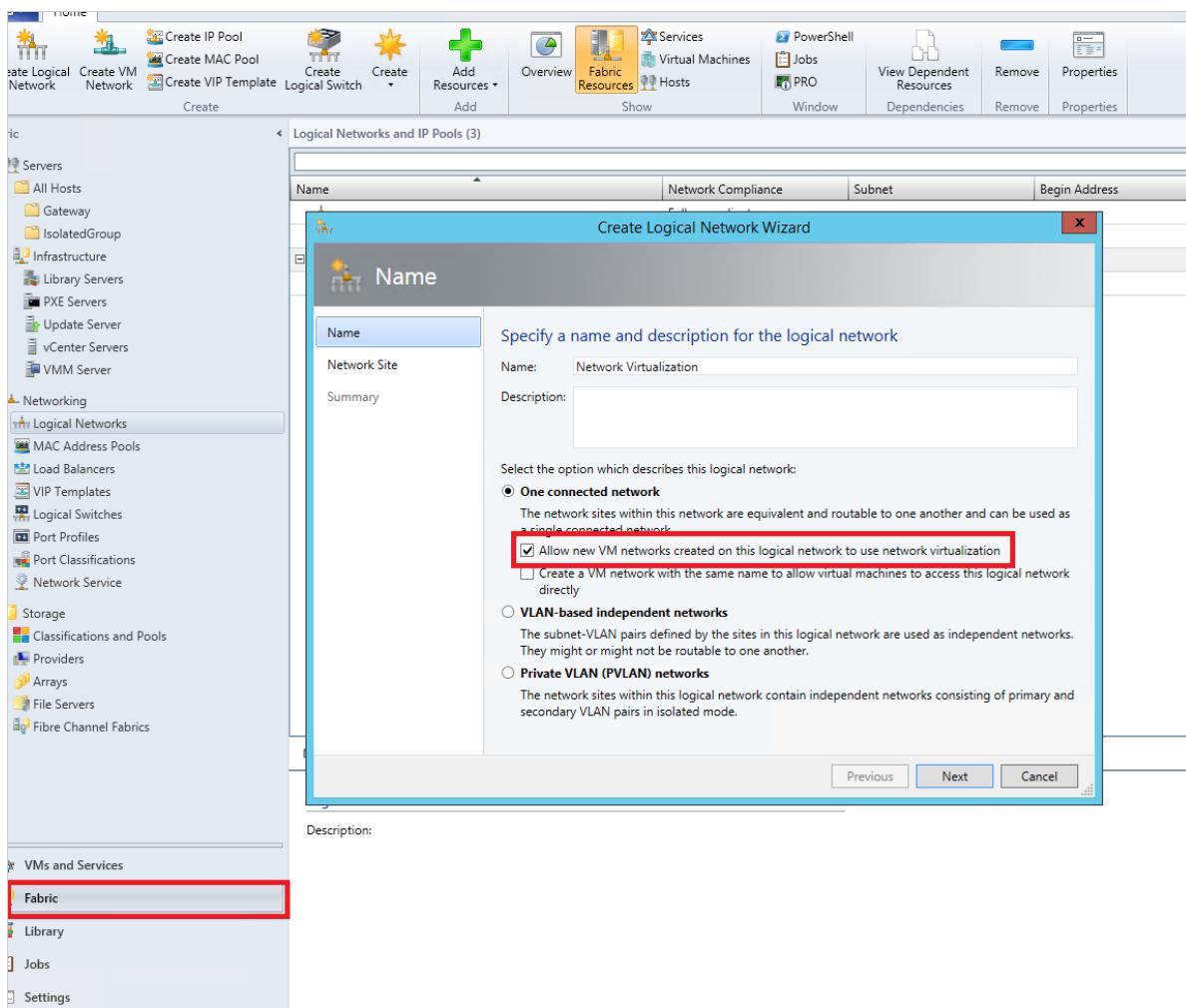
## Create logical networks

1. Log into the SCVMM admin console.

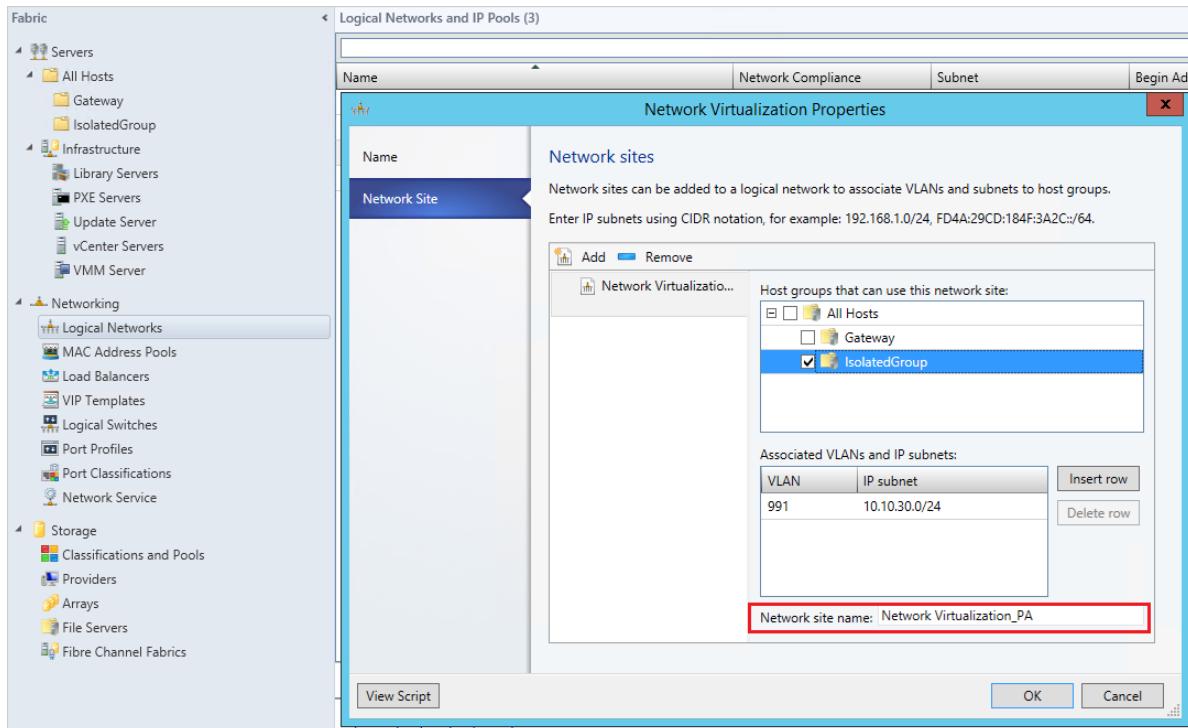
2. Go to **Fabric** -> **Networking** -> **Logical Networks** -> **Create new Logical Network**.



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**.

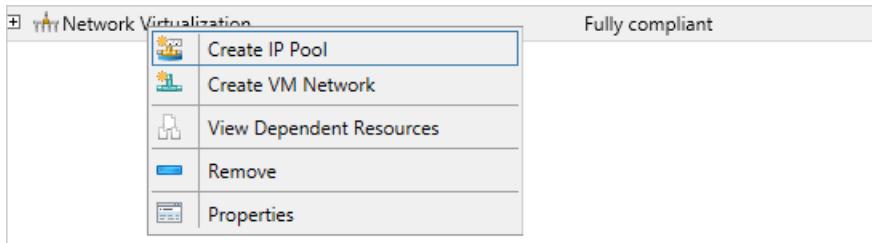


4. Add a new **Network Site** and select the host group to which the network site will be scoped. Enter the VLAN ID used to configure physical NIC in the Hyper-V host group and the corresponding routable IP subnet(s). To assist tracking, change the network site name to one that is memorable.

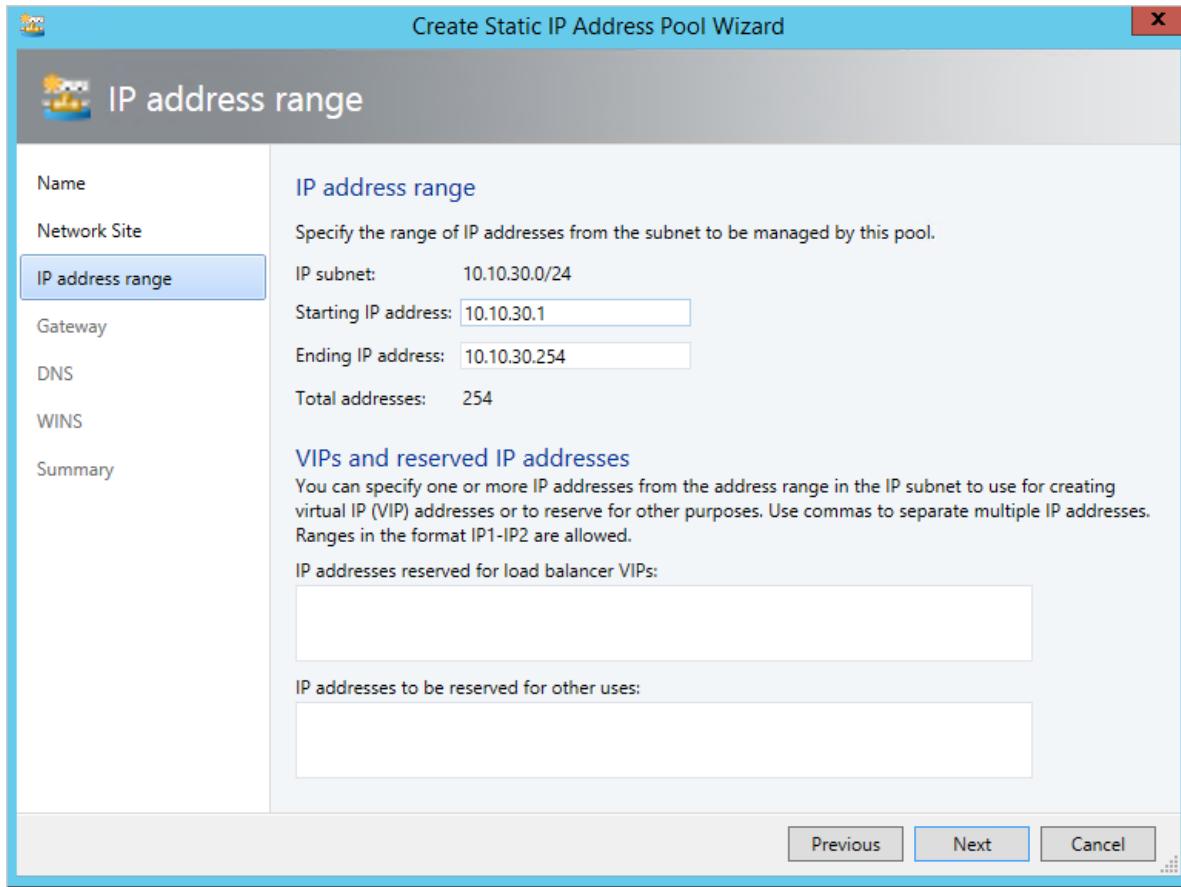


5. Click **Next** and **Save**.

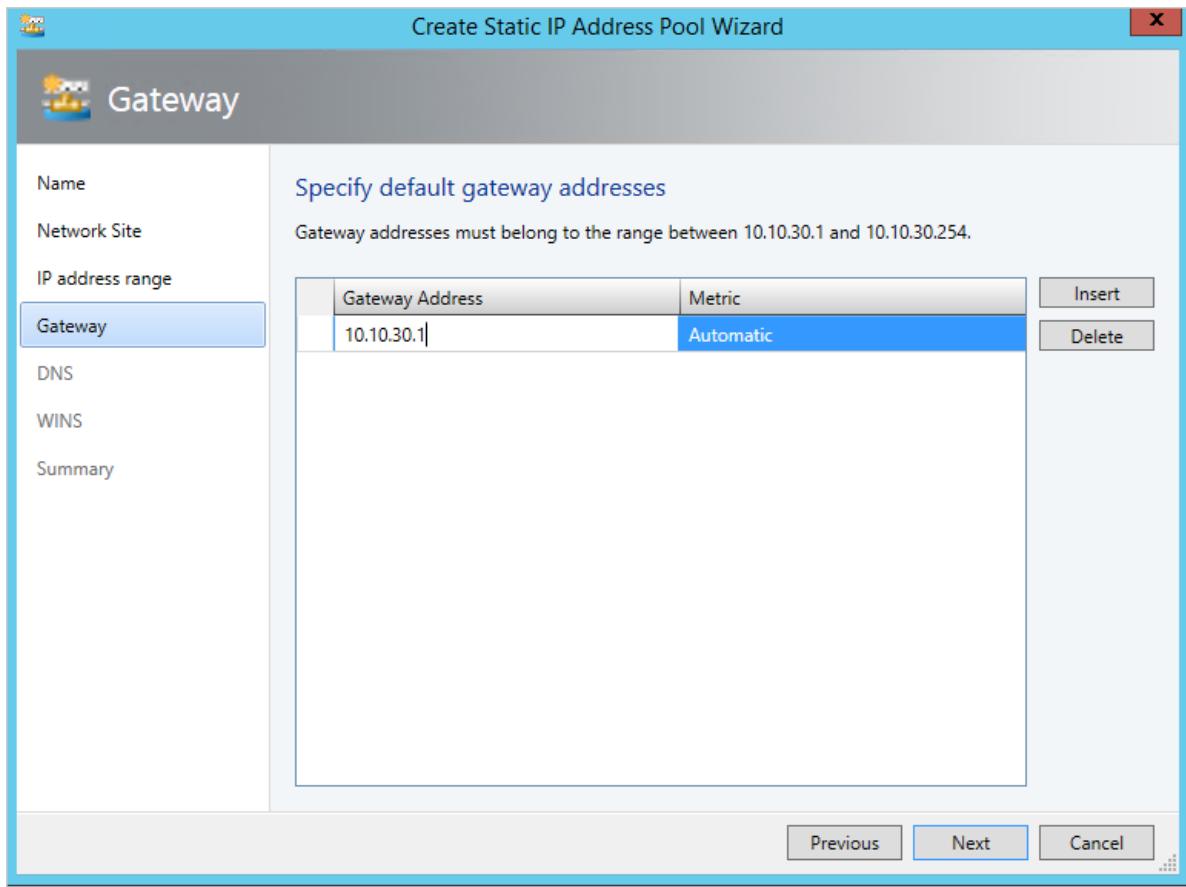
6. Create an IP pool for the new logical networks, enter a name, and click **Next**.



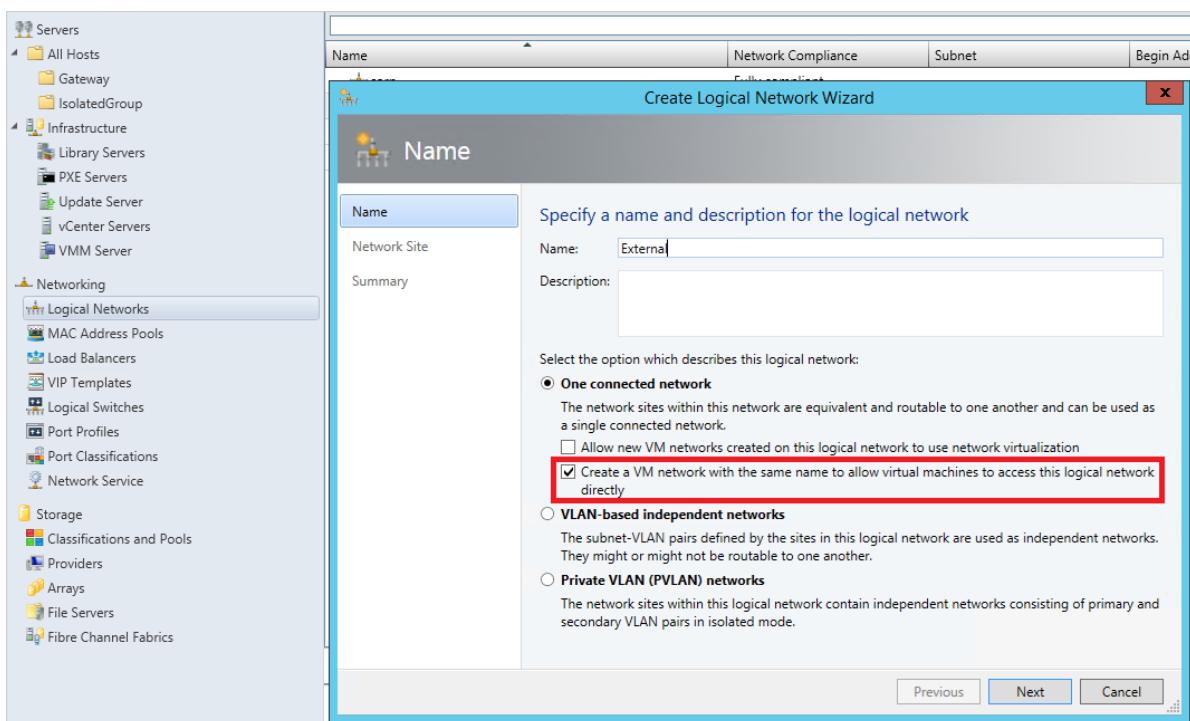
7. Select **Use and existing network site** and click **Next**. Enter the routable IP address range your network administrator configured for your VLAN and click **Next**. If you have multiple routable IP subnets associated with your VLAN, create an IP pool for each one.



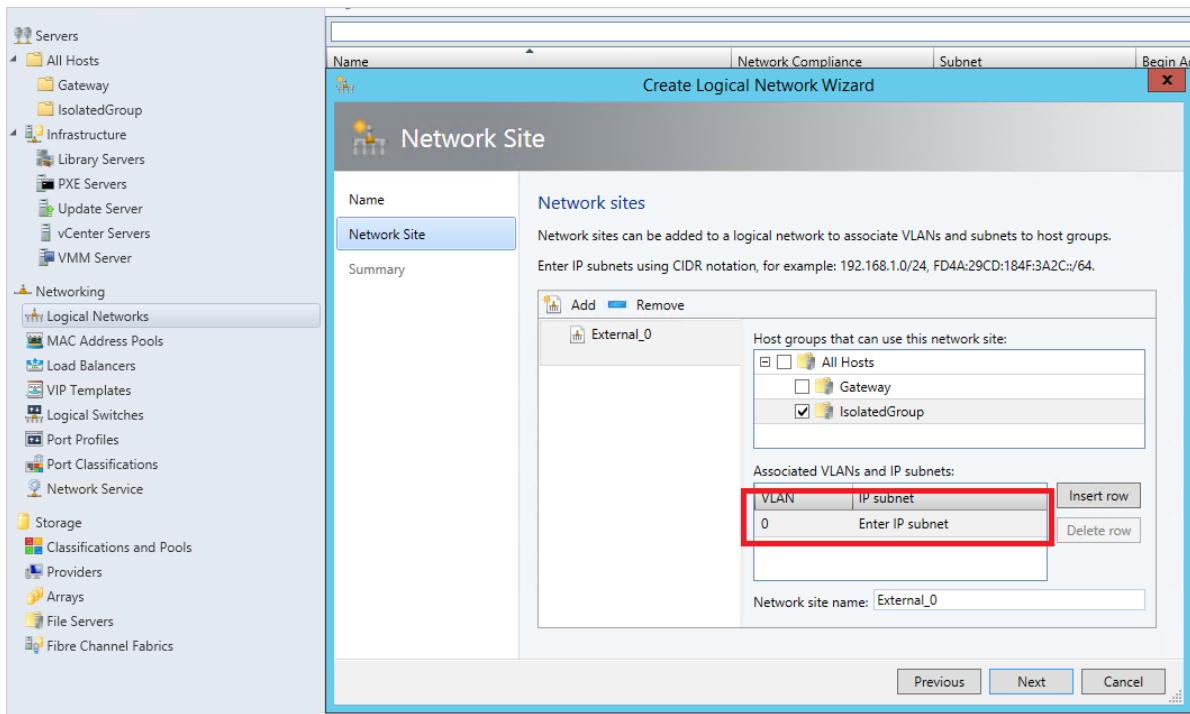
8. Provide the gateway address. By default, you can use the first IP address in your subnet.



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**.

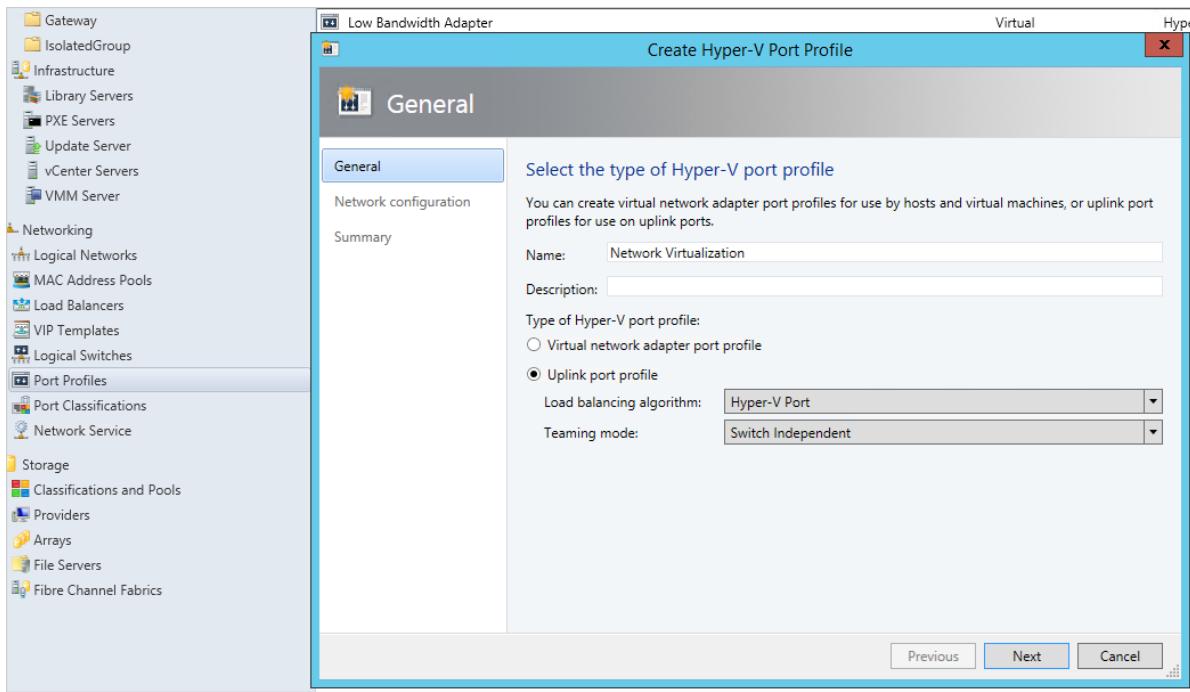
13. The result should look like the following in your administrator console after creating the logical networks.

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

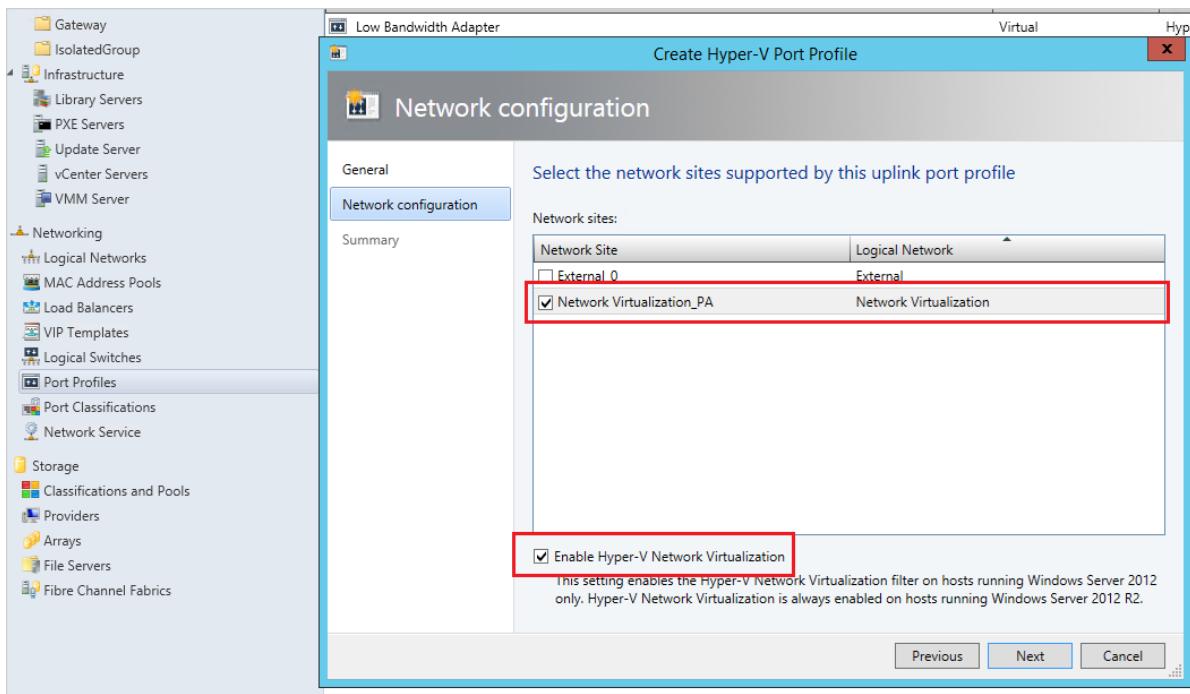
### Create port profiles

1. Go to **Fabric** -> **Networking** -> **Port profiles** and **Create Hyper-V port profile**.

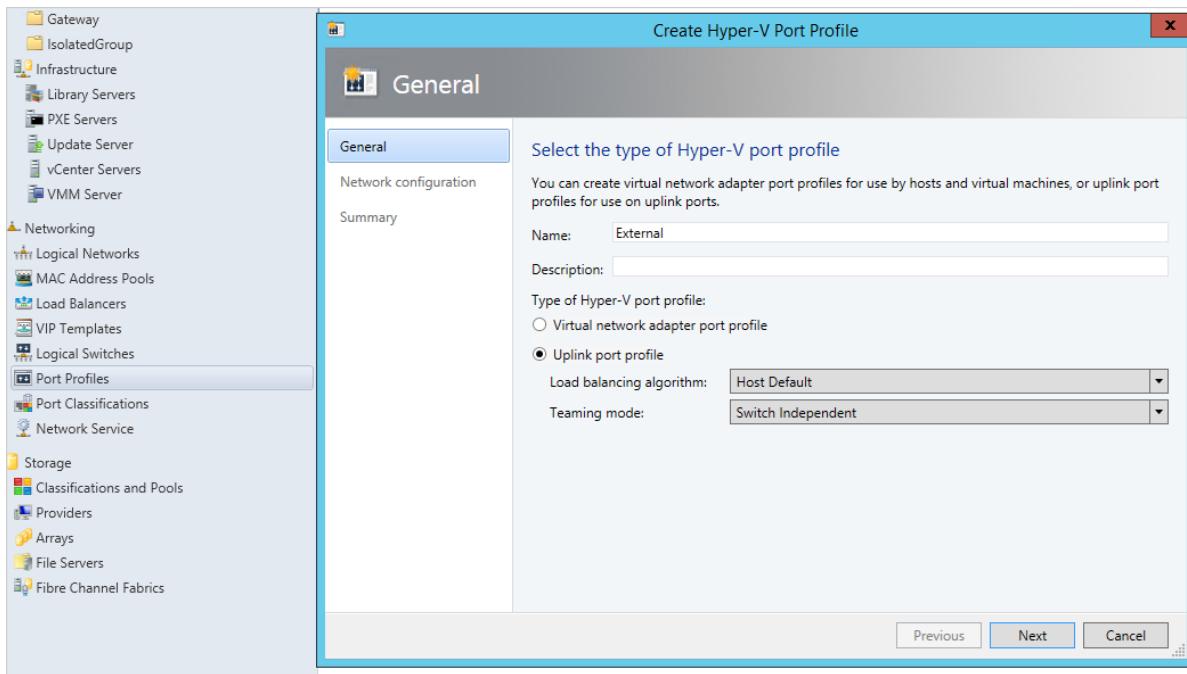
2. Select **Uplink port profile** and select **Hyper-V Port** as the load balancing algorithm, then click **Next**.



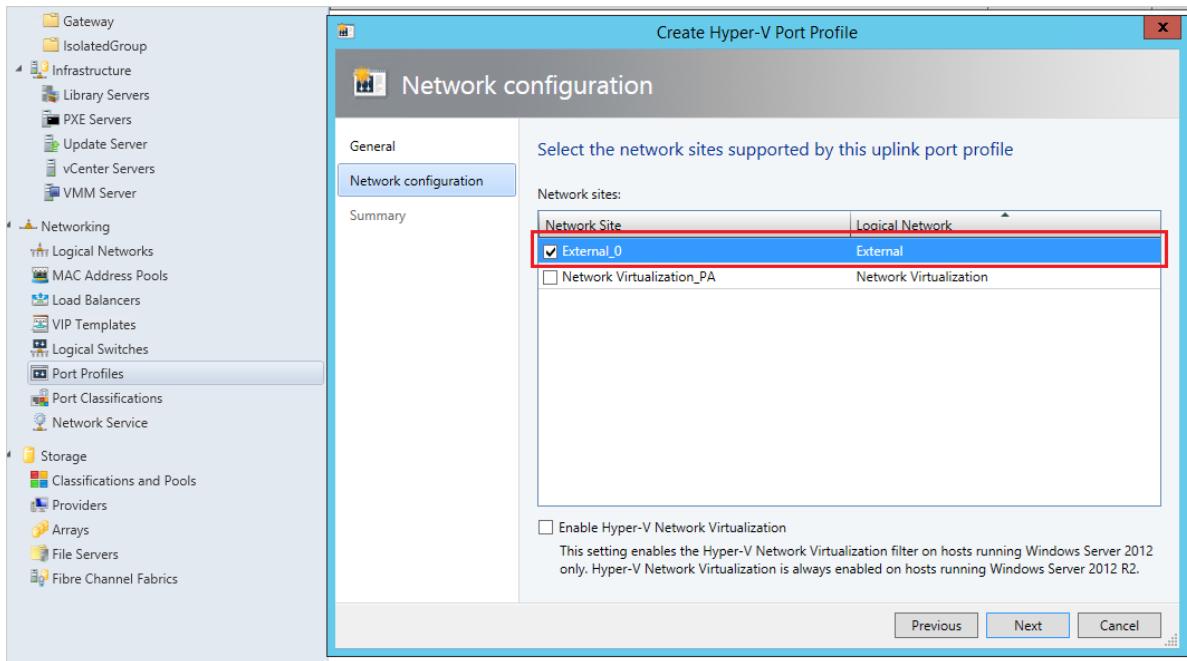
3. Select the Network Virtualization site created previously and choose the **Enable Hyper-V Network Virtualization** checkbox, then save the profile.



4. Now create another Hyper-V port profile for external logical network. Select **Uplink** mode and **Host default** as the load balancing algorithm, then click **Next**.

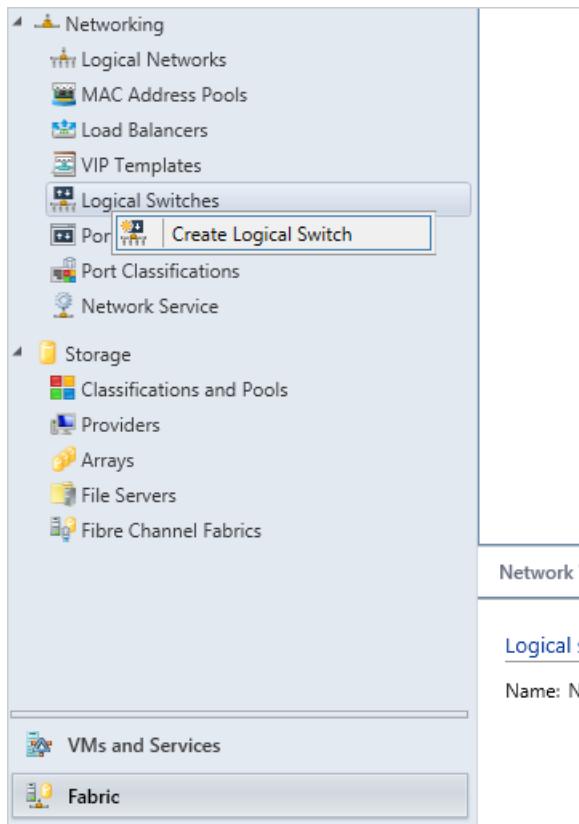


5. Select the other network site to be used for external communication, but and this time don't enable network virtualization. Then save the profile.

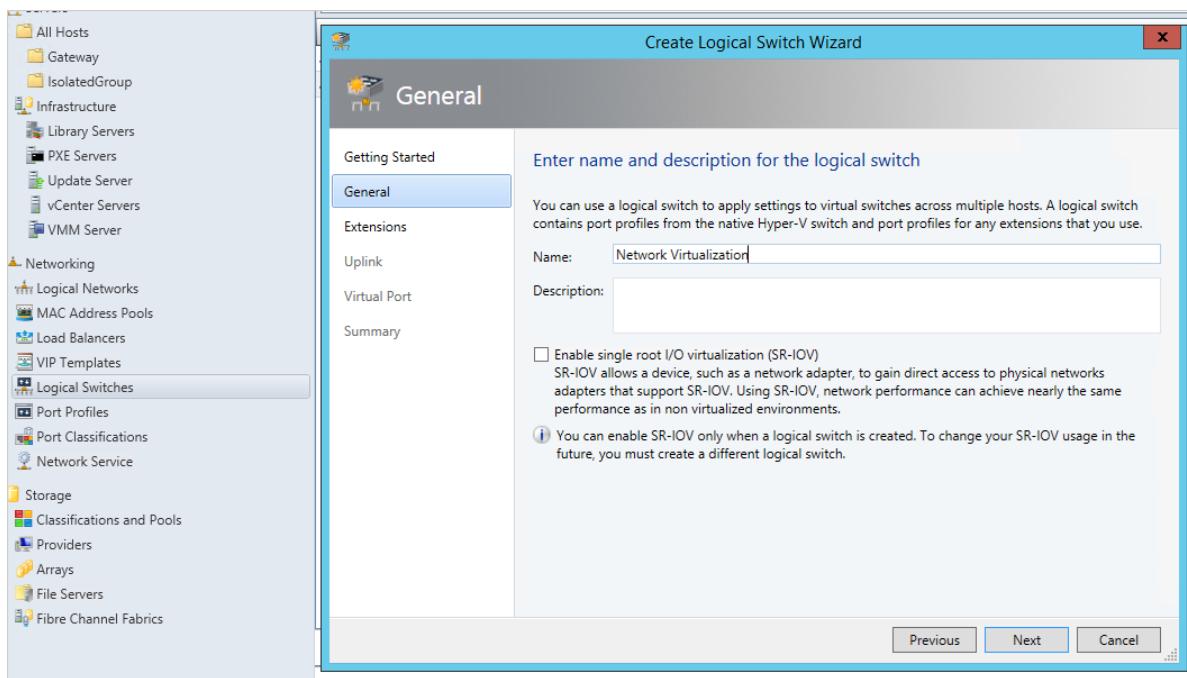


## Create logical switches

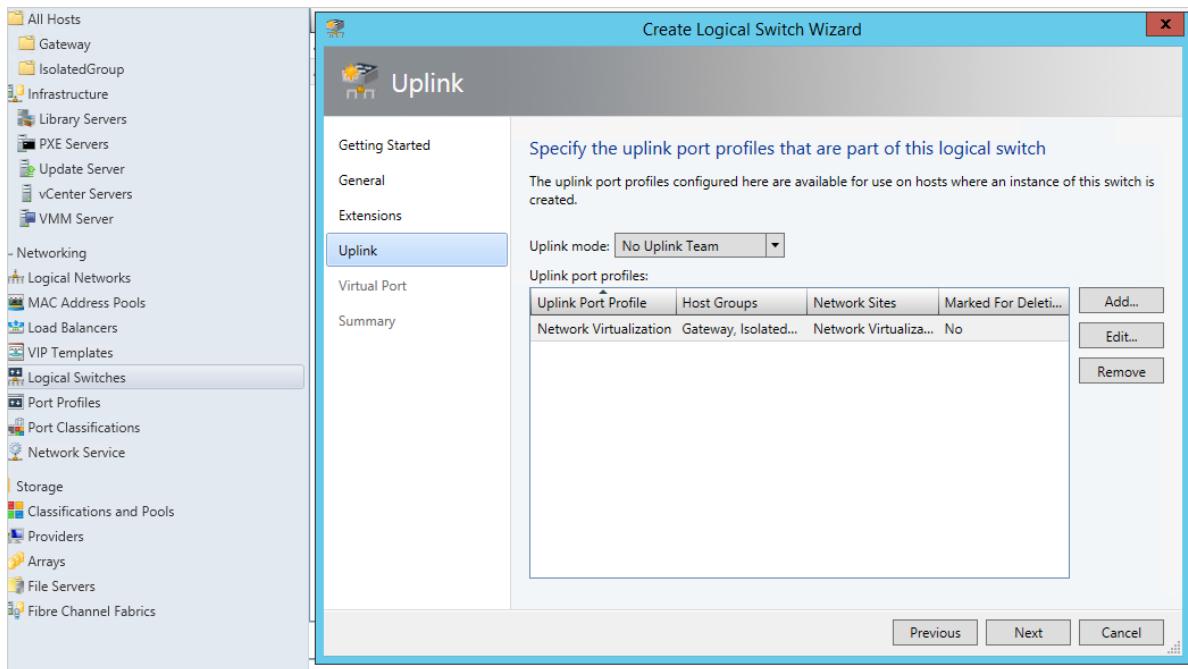
1. Go to **Fabric -> Networking -> Logical switches** and **Create Logical Switch**.



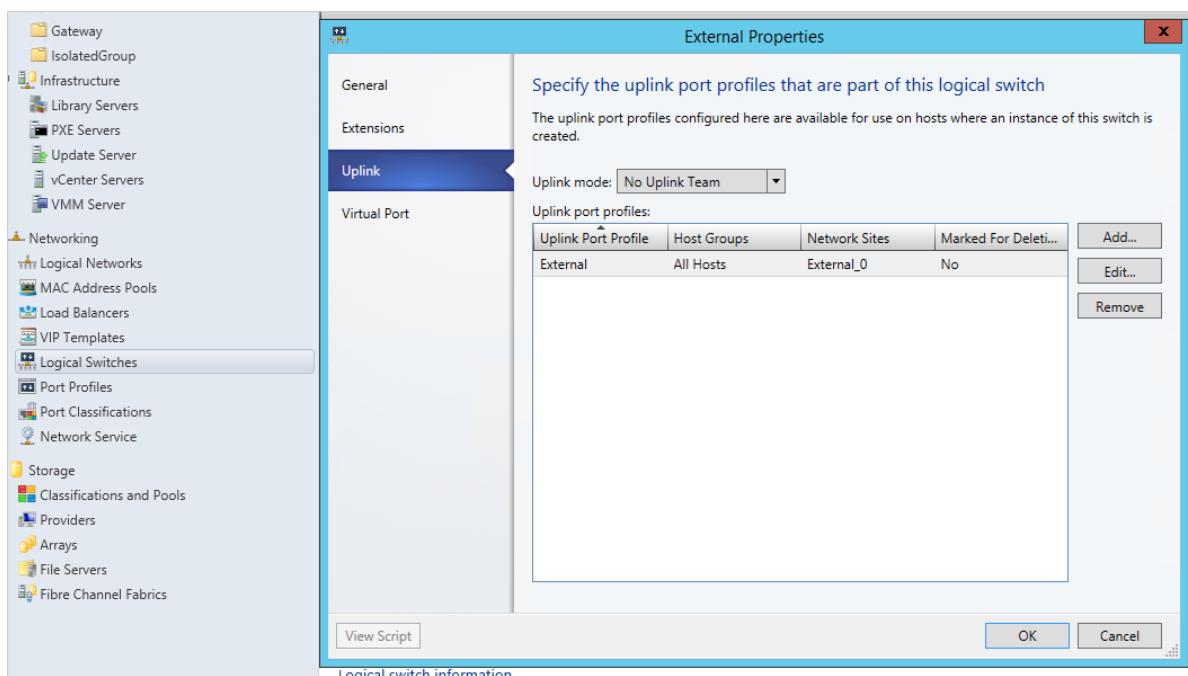
2. In the getting started wizard, click **Next** and enter a name for the switch, then click **Next**.



3. Click **Next** to open to **Uplink** tab. Click **Add uplink port profile** and add the network virtualization port profile you just created.

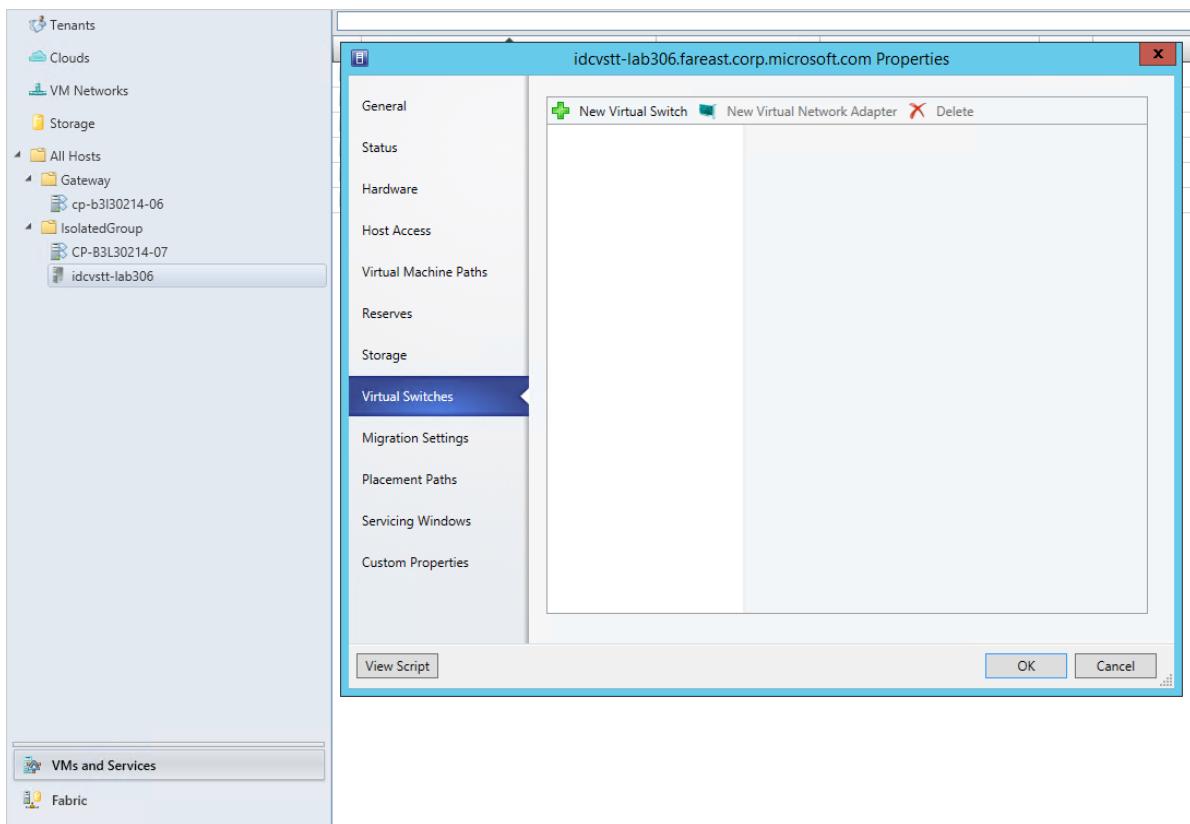


4. Click **Next** and save the logical switch.
5. Now create another logical switch for the external network for Internet communication. This time add the other uplink port profile you created for the external network.

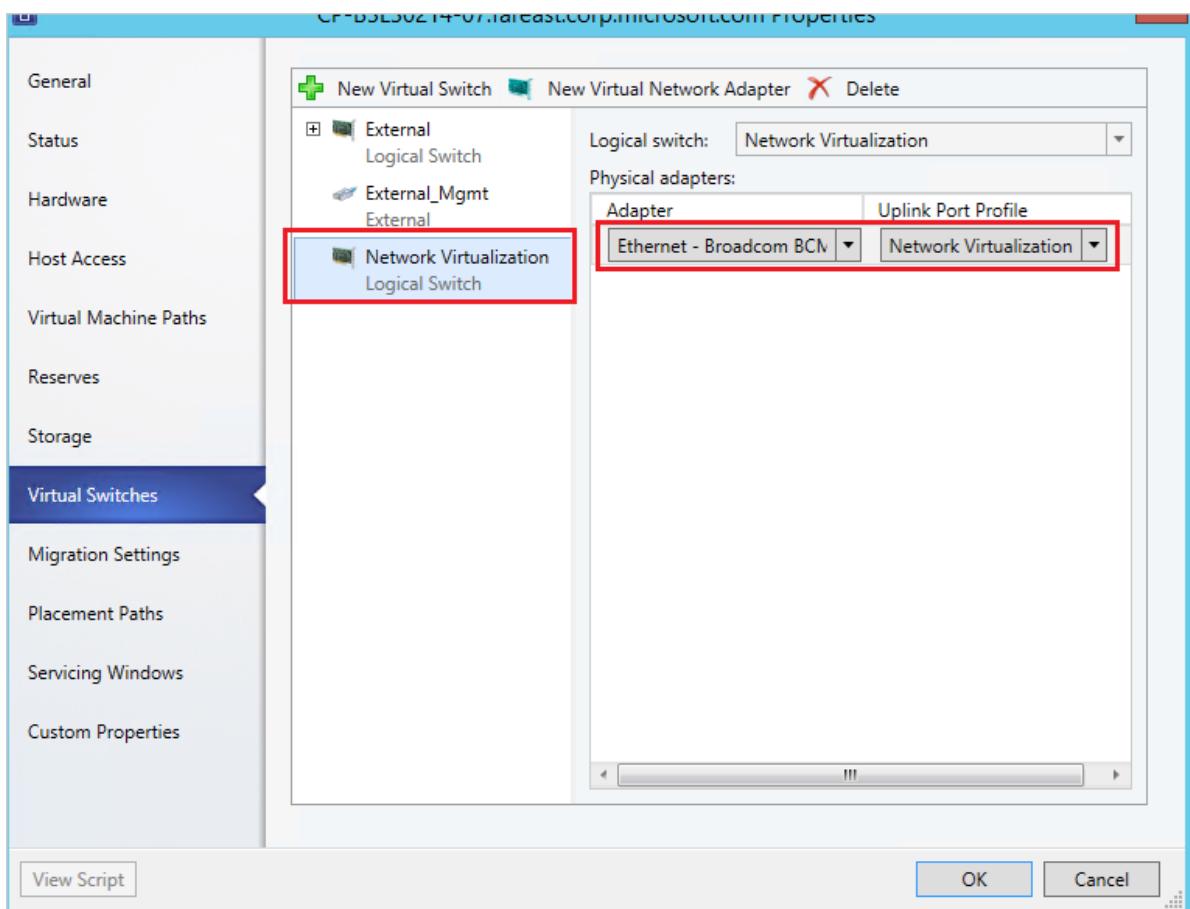


### Add logical switches to Hyper-V hosts

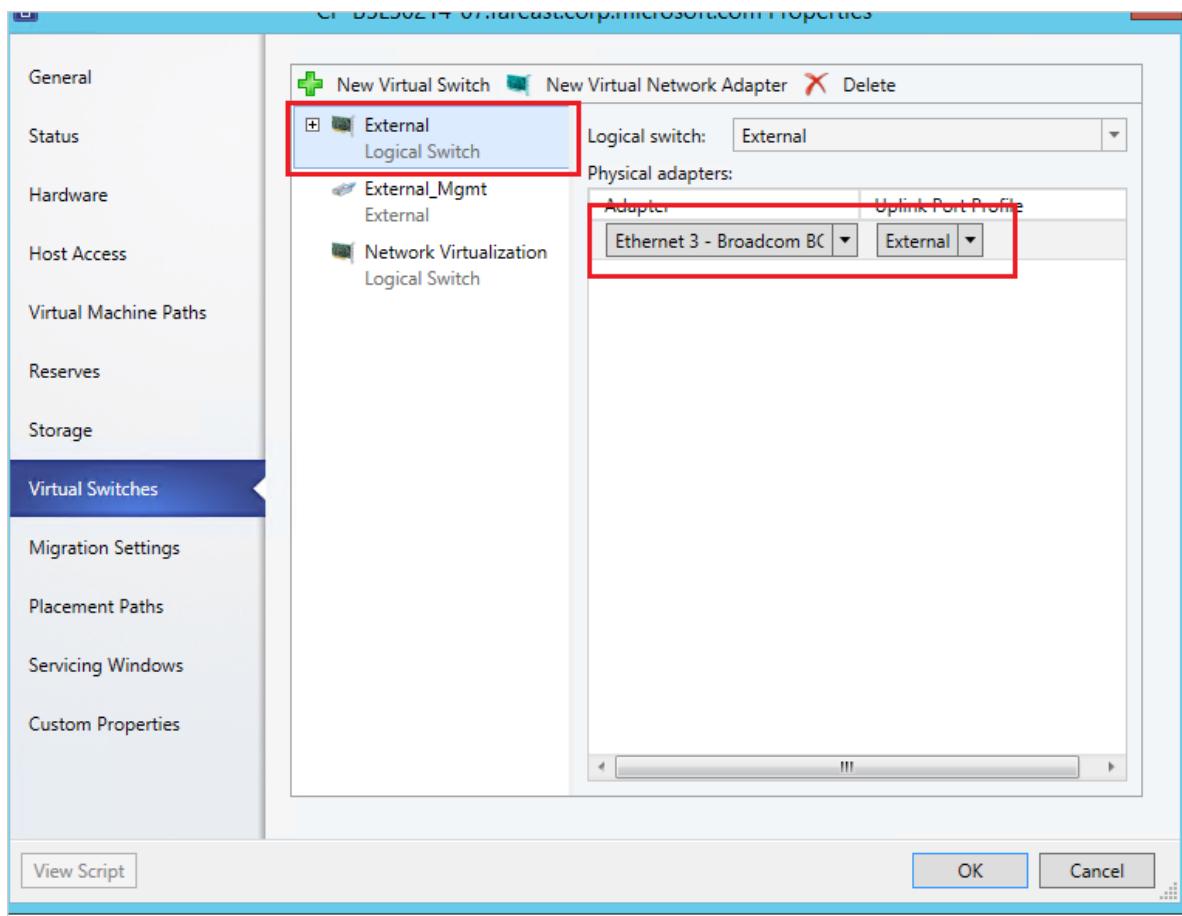
1. Go to **VM and Services** -> [Your host group] -> [each of the host machines in turn].
2. Right click and open the **Properties** -> **Virtual Switches** tab.



3. Click **New Virtual Switch** -> **New logical switch for network virtualization**. Assign the physical adapter you configured in trunk mode and select the network virtualization port profile.



4. Create another logical switch for external connectivity, assign the physical adapter used for external communication, and select the external port profile.



5. Do the same for all the Hyper-V hosts in the host group.

This is a one-time configuration for a specific host group of machines. After completing this setup, you can dynamically provision your isolated network of virtual machines using the **SCVMM extension** in TFS and VSTS builds and releases.

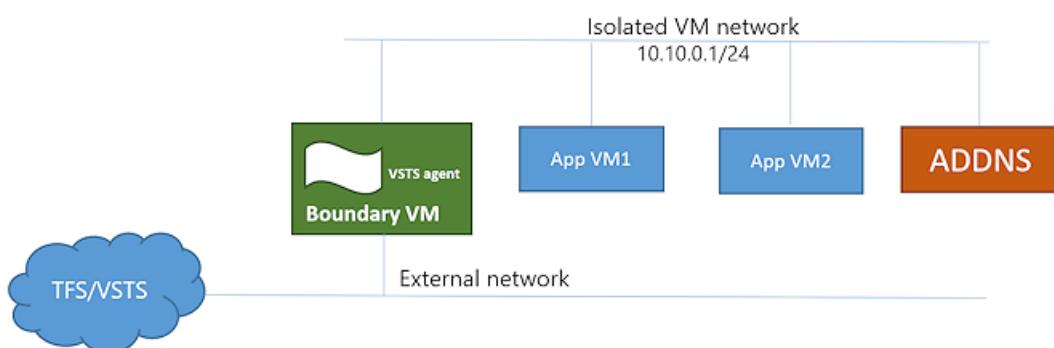
[Back to list of tasks](#)

## Create the required virtual network topology

Isolated virtual networks can be broadly classified into three topologies.

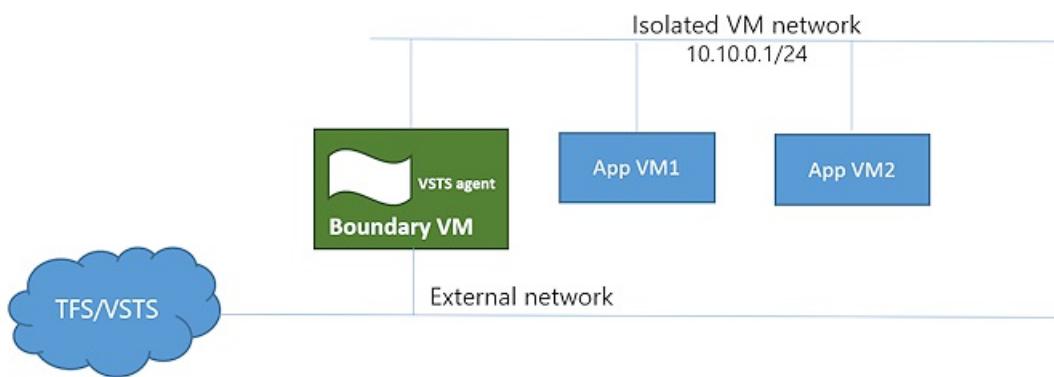
### Topology 1: AD-backed Isolated VMs

- A boundary VM with Internet/TFS connectivity.
- A VSTS/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.



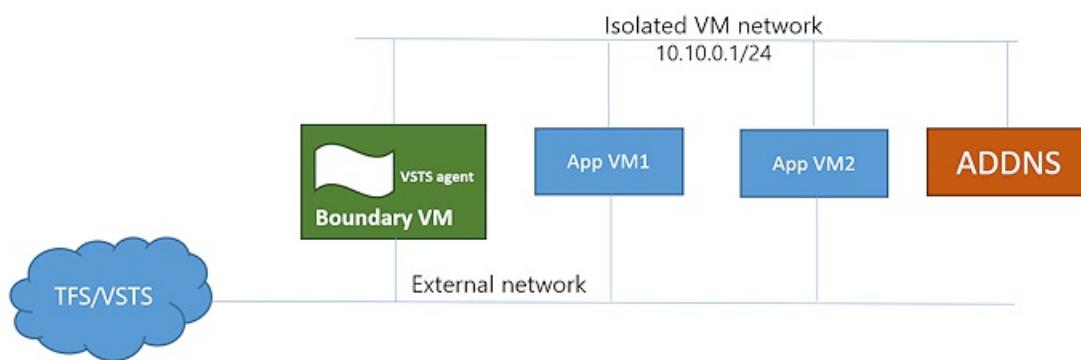
### Topology 2: Non-AD backed isolated VMs

- A boundary VM with Internet/TFS connectivity.
- A VSTS/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.
- A VSTS/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 VSTS instance and install the **SCVMM extension** if not already installed. For more information, see [Configure and deploy with SCVMM](#).

The **SCVMM task** provides a more efficient way capability to perform lab management operations using build and release definitions. You can manage SCVMM environments, provision isolated virtual networks, and implement build-deploy-test scenarios.

2. In a build or release definition, add a new **SCVMM** task.
3. In the **SCVMM** task, select an endpoint for the SCVMM server connection where you want to provision your virtual network and then select **New Virtual machines using Templates/Stored VMs and VHDS** to provision your VMs.

The screenshot shows the 'Tasks' tab selected in the pipeline editor. A task named 'SCVMM :NewVM action' is expanded. In the configuration pane, the 'Action' dropdown is highlighted with a red box and set to 'New Virtual Machine using Template/stored VM/VHD'.

4. You can create VMs from templates, stored VMs, and VHD/VHDx. Choose the appropriate option and enter the VM names and corresponding source information.

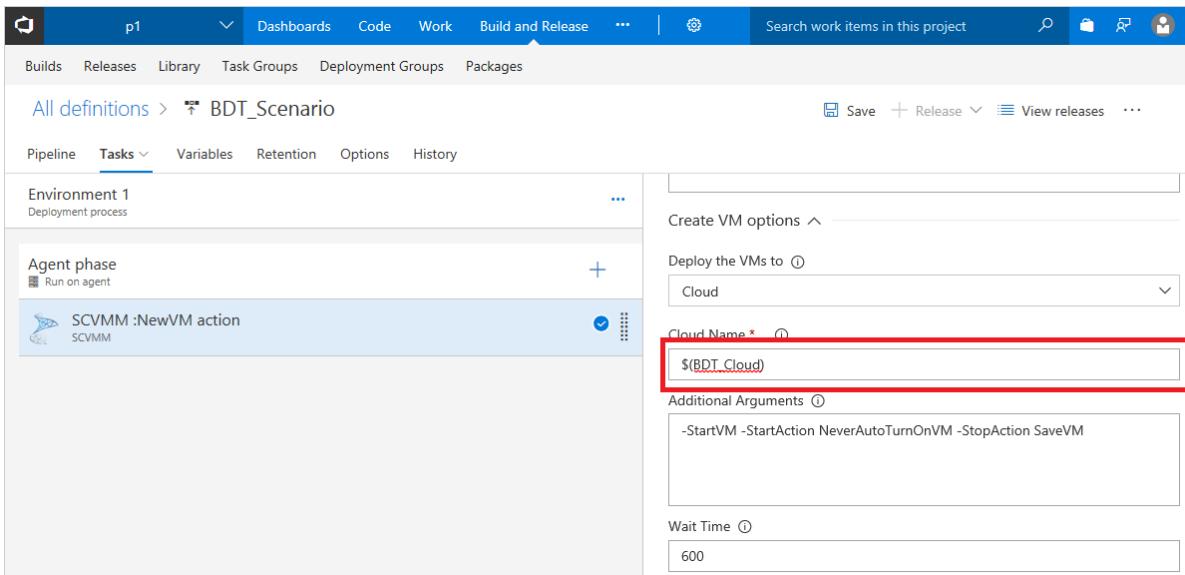
The screenshot shows the 'Tasks' tab selected in the pipeline editor. A task named 'SCVMM :NewVM action' is expanded. In the configuration pane, under 'VM Template options', the 'Create virtual machines from VM Templates' checkbox is checked. The 'Virtual machine names' field contains \$(AppVM1) and the 'VM template names' field contains \$(AppVM\_Template).

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.

The screenshot shows the 'Tasks' tab selected in the pipeline editor. A task named 'SCVMM :NewVM action' is expanded. In the configuration pane, under 'Stored VM options', the 'Create virtual machines from stored VMs' checkbox is checked. The 'VM names' field contains \$(AppVM2) and the 'Stored VMs' field contains \$(Stored\_AppVM).

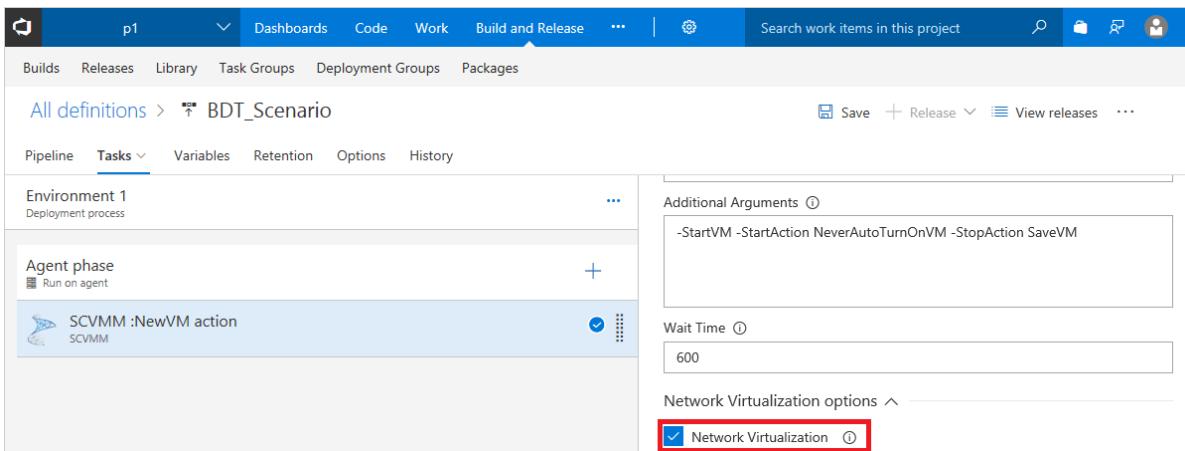
6. Enter the **Cloud Name** of the host where you want to provision your isolated network. In case of private cloud, ensure the host machines added to the cloud are connected to the same logical and external switches

as explained above.



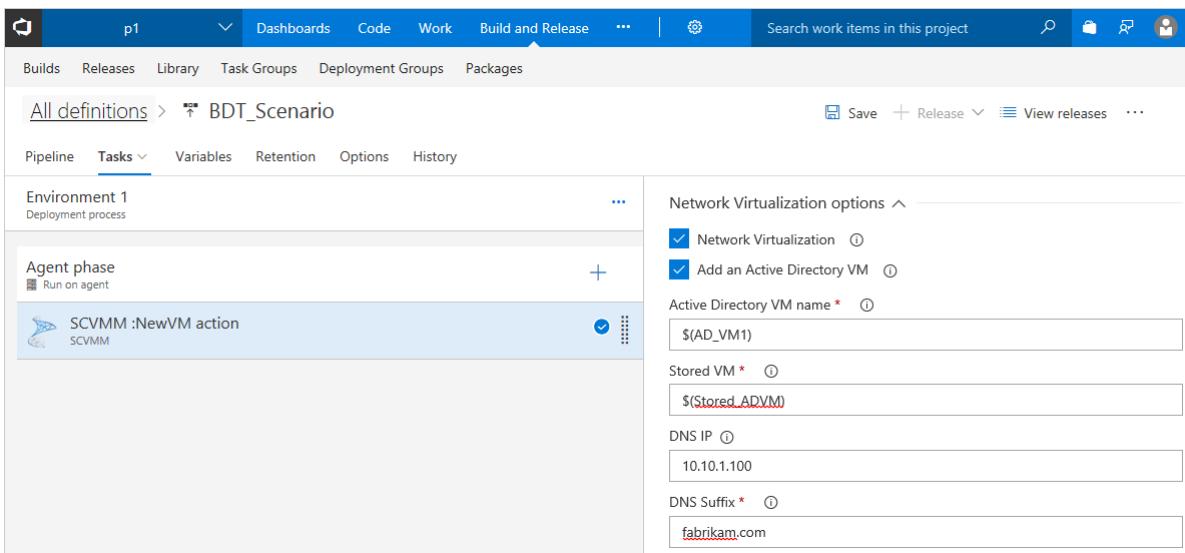
The screenshot shows the 'Tasks' tab selected in the pipeline editor. A task named 'SCVMM :NewVM action' is highlighted. On the right, there's a configuration panel for 'Create VM options'. The 'Cloud Name' field is explicitly labeled and contains the variable '\$(BDT\_Cloud)'. This field is highlighted with a red box.

7. Select the **Network Virtualization** option to create the virtualization layer.



The screenshot shows the same pipeline configuration as the previous step, but now the 'Network Virtualization' checkbox under 'Network Virtualization options' is checked and highlighted with a red box.

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.



The screenshot shows the final configuration of the 'SCVMM :NewVM action' task. The 'Network Virtualization' and 'Add an Active Directory VM' checkboxes are checked. The configuration panel on the right shows the following values:

- Active Directory VM name: \$(AD\_VM1)
- Stored VM: \$(Stored\_AdVm)
- DNS IP: 10.10.1.100
- DNS Suffix: fabrikam.com

9. Enter the settings for the **VM Network** and subnet you want to create, and the backing logical network you created in the [previous section](#) (Logical Networks). Ensure the VM network name is unique. If possible, append the release name for easier tracking later.

The screenshot shows the 'Tasks' tab selected in the pipeline editor. A red box highlights the 'VM Network' configuration section. It includes fields for 'Create new' (radio button selected), 'VM Network name' (\${VM\_Network}\_\${Release.DefinitionName}\_\${Release.ReleaseName}), 'VM network subnet' (10.10.1.0/24), and 'Logical Network name' (NetworkVirtualization).

10. In the **Boundary Virtual Machine options** section, set **Create boundary VM for communication with VSTS/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.

The screenshot shows the 'Tasks' tab selected in the pipeline editor. A red box highlights the 'Create boundary VM for VSTS/TFS communication' checkbox. Other fields shown include 'TFS/VSTS boundary VM name' (\$Boundary\_VM), 'TFS/VSTS boundary VM template' (\$Boundary\_Template), 'External VM network for boundary VM' (\$External\_Network), and 'Subnet name for external VM network'.

12. Provide details for configuring the boundary VM agent to communicate with TFS/VSTS. You can configure a deployment agent or an automation agent. This agent will be used for app deployments.

The screenshot shows the 'Tasks' tab of a pipeline definition named 'BDT\_Scenario'. Under the 'Agent phase' section, the 'Run on agent' icon is selected. In the configuration pane, the 'Boundary VM agent type' dropdown is set to 'Automation Pool agent', which is highlighted with a red box. Other fields include 'Pool name' set to '\$(AutomationPool)', 'Personal Access Token' set to '\$(MyPAT)', 'Agent name' set to '\$(MyAgent)\_\${Release.ReleaseName}', 'Agent installation path' set to '\$(AgentPath)', 'Boundary VM administrator' set to '\$(UserName)', and 'Boundary VM password' set to '\$(Password)'.

13. Ensure the agent name you provide is unique. This will be used as demand in succeeding phase 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 phase in your definition, after your network virtualization phase.
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 phase** or **Agent phase**.
3. In the phase 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 phase. In the case of deployment group phase, you must set the tag in the properties of the group machines.

The screenshot shows the 'Tasks' tab of the same pipeline definition. In the 'Agent phase' properties, a new demand is added for 'Agent.Name' with the condition 'equals' and the value '\$(MyAgent)\_\${Release.ReleaseName}'. This row is highlighted with a red box.

5. Inside the phase, add the tasks you require for deployment and testing.

The screenshot shows the Azure DevOps Pipeline editor interface. At the top, there's a navigation bar with 'p1' selected. Below it, a secondary navigation bar includes 'Builds', 'Releases', 'Library', 'Task Groups', 'Deployment Groups', and 'Packages'. The main area displays a 'Pipeline' view for a release definition named 'BDT\_Scenario'. The pipeline consists of an 'Environment 1' deployment process, which contains an 'Agent phase' task. This task is currently being edited, showing a 'PowerShell on Target Machines' configuration. The 'Display name' is set to 'Run PowerShell on \$(AppVM1), \$(AppVM2)'. The 'Machines' field, containing the value '\$(AppVM1), \$(AppVM2)', is highlighted with a red box. Other fields shown include 'Admin Login' and '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 definition. 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 the **Build & Release** hub.

[Back to list of tasks](#)

## See also

- [Configure and deploy with SCVMM](#)
- [Task actions for managing VMs using SCVMM](#)
- [Hyper-V Network Virtualization Overview](#)

## Q&A

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Set build and release permissions

2/16/2018 • 5 min to read • [Edit Online](#)

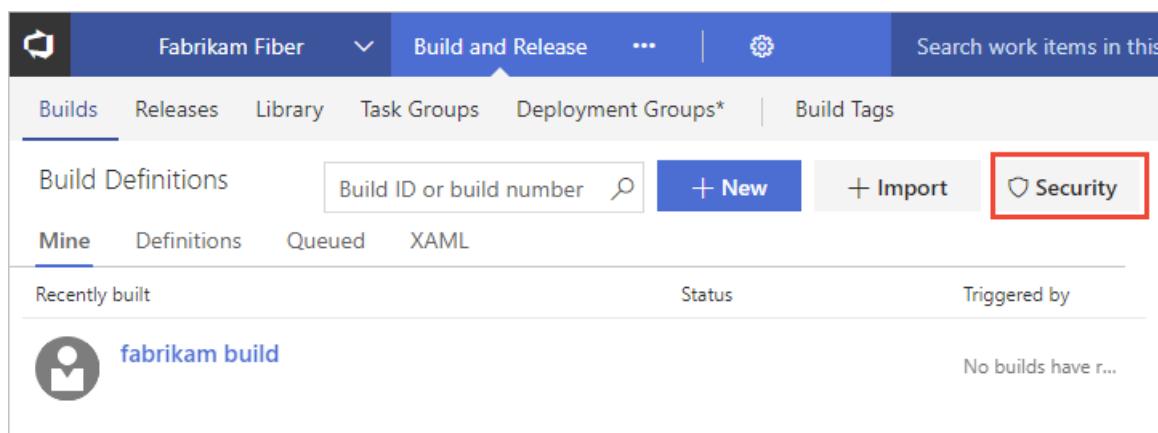
**VSTS | TFS 2018 | TFS 2017 | TFS 2015 | TFS 2013**

Permissions for build and release functions are primarily set at the object-level for a specific build or release, or for select tasks, at the collection level. For a simplified view of permissions assigned to built-in groups, see [Permissions and access](#).

In addition to permission assignments, you manage security for several resources—such as variable groups, secure files, and deployment groups—by adding users or groups to a role. You grant or restrict permissions by setting the [permission state to Allow or Deny](#), either for a security group or an individual user. For definitions of each build and release permission and role, see [Build and release permissions](#).

## Set permissions for build definitions

1. To set the permissions for all build definitions, click the Security From the web portal **Build+Release** hub, **Builds** page



The screenshot shows the VSTS Build+Release hub with the 'Builds' tab selected. At the top, there's a navigation bar with 'Fabrikam Fiber' and 'Build and Release' options. Below the navigation is a toolbar with 'Build ID or build number' search, '+ New', '+ Import', and a 'Security' button, which is highlighted with a red box. The main area displays a table for 'Build Definitions'. The first row shows a build named 'fabrikam build' with status 'Recently built' and trigger 'No builds have r...'. There are tabs for 'Mine', 'Definitions', 'Queued', and 'XAML'.

To set the permissions for a specific build definition, open the context menu for the build and click Security.

The screenshot shows the Microsoft DevOps interface for managing build definitions. At the top, there's a navigation bar with links for Dashboards, Code, Work, Build & Release, Test, and Wiki\*. Below that is a secondary navigation bar with tabs for Builds, Releases, Library, Task Groups, Deployment Groups\*, and Build Tags. The 'Builds' tab is selected.

The main area is titled 'Build Definitions' and has filters for Mine, All Definitions, Queued, and XAML. It lists a single item: 'Recently built' with the name 'fabrikam build'. To the right of this item is a 'Status' column. A context menu is open over the 'fabrikam build' item, showing options like Queue new build..., Edit..., View definition summary, Add to my favorites, Add to team favorites, Clone..., Export, Rename..., Save as a template..., Delete definition, Security... (which is highlighted with a red box), and Add to dashboard.

- Choose the group you want to set permissions for, and then change the permission setting to Allow or Deny.

For example, here we change the permission for Edit build definition for the Contributors group to Allow.

Permissions for fabrikam build

[+ Add...](#)   [Inheritance ▾](#)

Search

- ▼ VSTS Groups
  - [Build Administrators](#)
  - [Contributors](#)
  - [Fabrikam Fiber Team](#)
  - [Project Administrators](#)
  - [Readers](#)
  - [Project Collection Administrators](#)
  - [Project Collection Build Administrators](#)
  - [Project Collection Build Service Accounts](#)
  - [Project Collection Test Service Accounts](#)
- ▼ Users
  - [Fabrikam Fiber Build Service \(kelliott\)](#)
  - [Project Collection Build Service \(kelliott\)](#)

**ACCESS CONTROL SUMMARY**  
Shows information about the permissions being granted to this identity

Administer build permissions	Not set
Delete build definition	Not set
Delete builds	Not set
Destroy builds	Not set
Edit build definition	<b>Allow</b>
Edit build quality	Allow (inherited)
Manage build qualities	Not set
Manage build queue	Not set
Override check-in validation by build	Not set
Queue builds	Allow (inherited)
Retain indefinitely	Not set
Stop builds	Not set
Update build information	Not set
View build definition	Allow (inherited)
View builds	Allow (inherited)

[Clear explicit permissions](#)

[Remove](#)   [Save changes](#)   [Undo changes](#)

[Close](#)

3. Save your changes.

## Set permissions for release definitions

- From the web portal **Build-Release** hub, **Releases** page, open the Security dialog for all release definitions.

The screenshot shows the VSTS Build-Release hub with the 'Releases' tab selected. In the main area, there is a list of 'Release Definitions' with one named 'All release definitions' currently selected. A context menu is open over this selection, and the 'Security...' option is highlighted with a blue box. The menu also includes options like 'Lock', 'Unlock', and 'Title'. At the bottom of the menu, there is a 'Release Definition' section.

If you want to manage the permissions for a specific release, then open the Security dialog for that release.

- Choose the group you want to set permissions for, and then change the permission setting to Allow or Deny.

For example, here we deny access to several permissions for the Contributors group.

Permission	Setting
Administer release permissions	Not set
Create releases	Allow
Delete release definition	Allow
Delete release environment	Allow
Delete releases	Allow
Edit release definition	Deny
Edit release environment	Allow
Manage deployments	Deny
Manage release approvers	Allow
Manage releases	Deny
View release definition	Allow
View releases	Allow

- Save your changes.

## Manage Library roles for variable groups, secure files, and deployment groups

Permissions for [variable groups](#), [secure files](#), and [deployment groups](#) are managed by roles. For a description of the roles, see [About security roles](#).

### NOTE

**Feature availability:** These features are available on VSTS and TFS 2017 and later versions.

You can set the security for all artifacts for a team project, as well as set the security for individual artifacts. The method is similar for all three artifact types. You set the security for variable groups and secure files from the **Build and Release** hub, **Library** page, and for deployment groups, from the **Deployment groups** page.

For example, here we show how to set the security for variable groups.

1. **Build-Release** hub, **Library** page, open the Security dialog for all variable groups.

The screenshot shows the Azure DevOps interface with the 'Library' tab selected. At the top right, there is a blue button labeled '+ Variable group' and a red-bordered button labeled 'Security'. Below the buttons, there are two links: 'Variable groups' (underlined) and 'Secure files'.

If you want to manage the permissions for a specific variable group, then open the Security dialog for that group.

This screenshot is similar to the first one, but it shows a context menu for a variable group named 'FF group'. The menu includes options for 'Edit', 'Delete', and 'Security'. The 'Security' option is highlighted with a red box.

2. Add the user or group and choose the role you want them to have.

For example, here we deny access to several permissions for the Contributors group.

The screenshot shows the 'Assign security roles for Library' dialog. A red arrow points from the 'Add' button in the main dialog to a sub-dialog titled 'Add user'. In the sub-dialog, a user named 'Raisa Pokrovskaya' is selected, and the 'Role' dropdown is set to 'User'. A note at the bottom states: 'User can use, but cannot manage the library items.' There are 'Add' and 'Close' buttons at the bottom of the sub-dialog.

3. Click **Add**.

## Manage task group permissions

Permissions for task groups are subject to a hierarchical model. You use task groups to encapsulate a sequence of tasks already defined in a build or a release definition into a single reusable task. You [define and manage task groups](#) in the **Task groups** tab of the **Build and Release** hub.

**NOTE**

**Feature availability:** These features are available on VSTS and TFS 2017 and later versions.

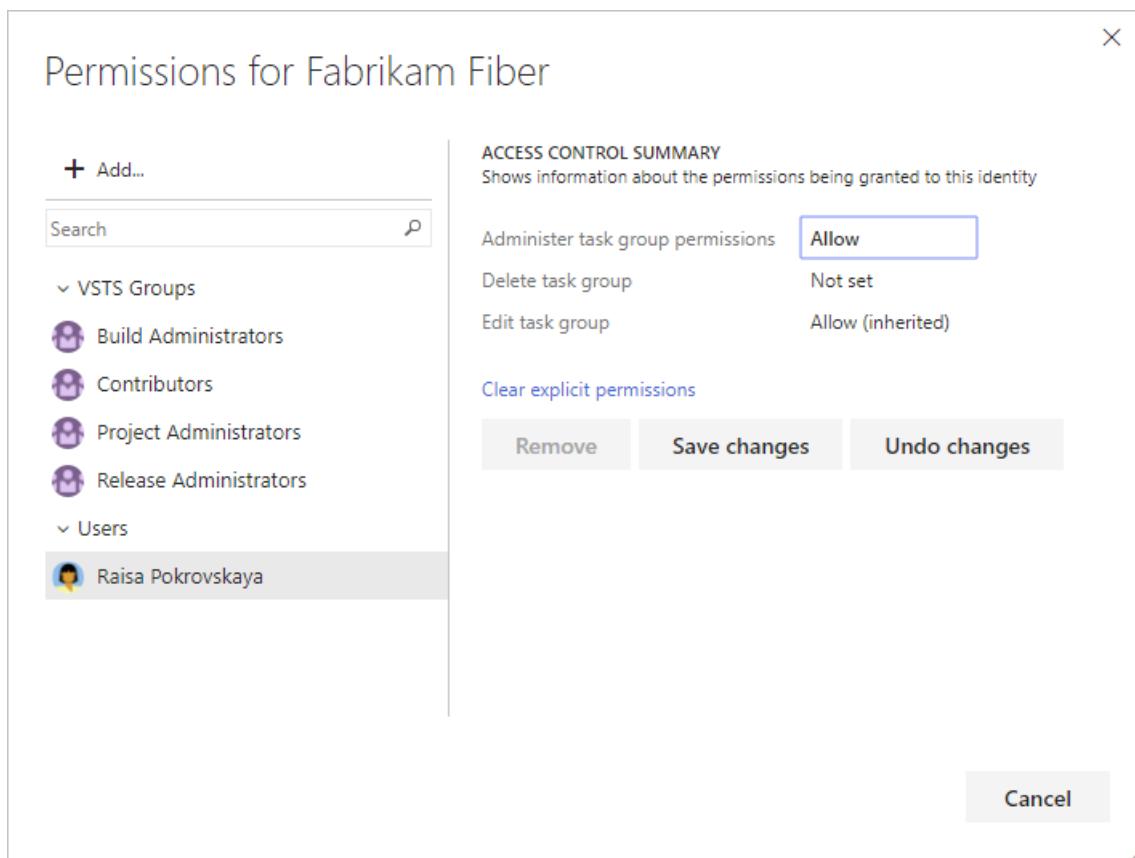
1. From the web portal **Build-Release** hub, **Task groups** page, open the Security dialog for all task groups.

![Open the Security dialog for all task groups](\_img/set-build-release-permissions/open-task-group-all-security.png)

If you want to manage the permissions for a specific task group, then open the Security dialog for that group.

1. Add the user or group and then set the permissions you want them to have.

For example, here we add Raisa and set her permissions to Administer all task groups.



2. Click **Add**.

## Set collection-level permissions to administer build resources

1. From the web portal user context, open the admin context by clicking the gear icon and choosing **Account settings** or **Collection settings**.
2. Click **Security**, and then choose the group whose permissions you want to modify.

Here we choose the Build Administrators group and change the **Use build resources** permission. For a description of each permissions, see [Permissions and groups reference](#), [Collection-level permissions](#).

fabrikam > Project Collection Build Administrator

**Permissions** Members Member of

Members of this group should include accounts for people who should be able to administer the build resources.

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

[Clear explicit permissions](#)

**Save changes** **Undo changes**

3. Save your changes.

## Manage permissions for agent queues and service endpoints

You manage the security for [agent pools](#) and [service endpoints](#) by adding users or groups to a role. The method is similar for both agent queues and service endpoints. You will need to be a member of the Project Administrator group to manage the security for these resources.

### NOTE

**Feature availability:** These features are available on VSTS and TFS 2015 and later versions.

For example, here we show how to add a user to the Administrator role for a service endpoint.

1. From the web portal, click the gear icon to open the project settings admin context.
2. Click **Services**, click the service endpoint that you want to manage, and then click **Roles**.

3. Add the user or group and choose the role you want them to have. For a description of each role, see [About security roles](#).

For example, here we add Raisa to the Administrator role.

4. Click **Add**.

## Manage permissions for agent pools and deployment pools

You manage the security for [agent pools](#) and [deployment pools](#) by adding users or groups to a role. The method is similar for both types of pools.

### NOTE

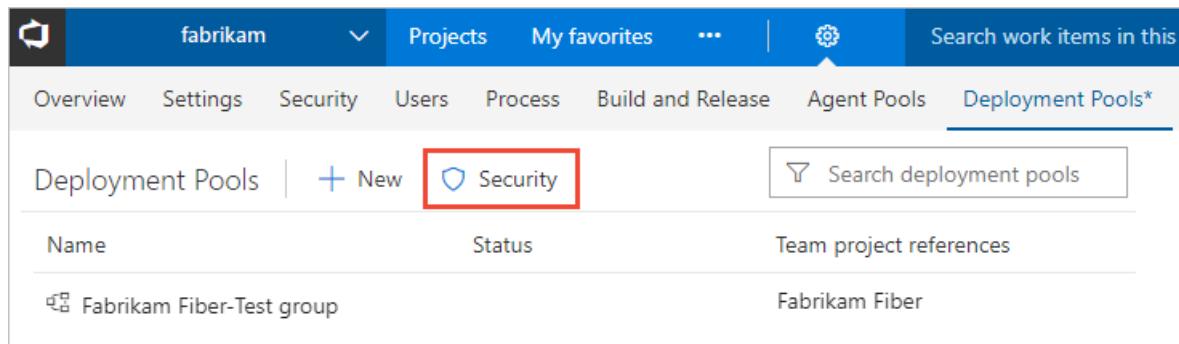
**Feature availability:** These features are available on VSTS and TFS 2018 and later versions.

You will need to be a member of the Project Collection Administrator group to manage the security for a pool. Once you've been added to the Administrator role, you can then manage the pool. For a description of each role, see [About security roles](#).

1. From the web portal, click the gear icon and choose Account settings or Collection settings to

open the collection-level settings admin context.

2. Click **Deployment Pools**, and then open the **Security** dialog for all deployment pools.

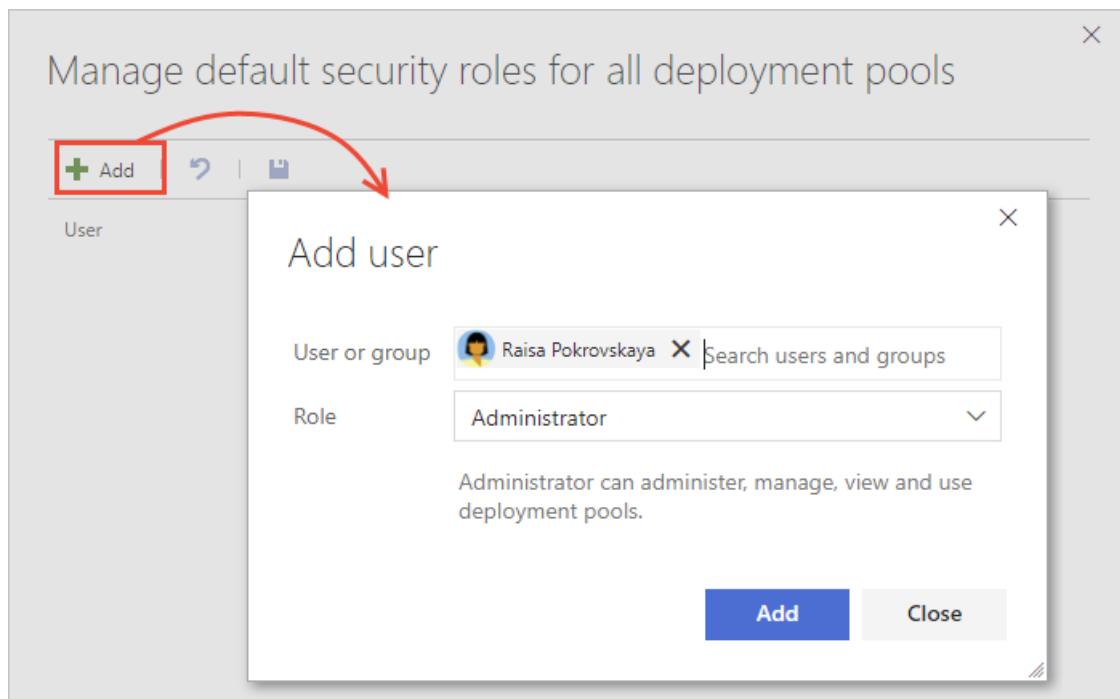


The screenshot shows the 'Deployment Pools' section of the Azure DevOps interface. At the top, there's a navigation bar with 'fabrikam' and dropdown menus for 'Projects', 'My favorites', and 'Deployment Pools\*'. Below this is a sub-navigation bar with 'Overview', 'Settings', 'Security', 'Users', 'Process', 'Build and Release', 'Agent Pools', and 'Deployment Pools\*'. The 'Deployment Pools\*' item is underlined. A red box highlights the 'Security' button. To the right of the sub-navigation is a search bar labeled 'Search work items in this collection'. Below the sub-navigation, there are two columns: 'Name' and 'Status'. Under 'Name', there's a row for 'Fabrikam Fiber-Test group' with 'Fabrikam Fiber' under 'Team project references'. A search bar 'Search deployment pools' is also present.

If you want to manage the permissions for a specific deployment group, then open the Security dialog for that group.

3. Add the user or group and choose the role you want them to have.

For example, here we add Raisa to the Administrator role.



The screenshot shows the 'Add user' dialog box. It has a header 'Add user' and a sub-header 'User'. It contains fields for 'User or group' (with 'Raisa Pokrovskaya' selected) and 'Role' (set to 'Administrator'). A descriptive note below says 'Administrator can administer, manage, view and use deployment pools.' At the bottom are 'Add' and 'Close' buttons. A red box highlights the 'Add' button in the main 'Manage default security roles' page, and a red arrow points from it to the 'Add' button in the dialog box.

4. Click **Add**.

## Related notes

### Default build and release permissions

- [Default permissions and access](#)
- [Permissions and groups reference](#)

# Troubleshooting build

1/29/2018 • 9 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

## Run commands locally at the command prompt

It is helpful to narrow whether a build failure is the result of a TFS/VSTS product issue (agent or tasks). Build failures may also result from external commands.

Check the build log for the exact command-line executed by the failing step. 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 build machine and run the command as the service account.

For example, is the problem happening during the MSBuild part of your build process (for example, are you using either the [MSBuild](#) or [Visual Studio Build](#) step)? If so, then try running the same [MSBuild command](#) on a local machine using the same arguments. If you can reproduce the problem on a local machine, then your next steps are to investigate the [MSBuild](#) problem.

### Differences between local command prompt and agent

Keep in mind, some differences are in effect when executing a command on a local machine and when a build is running on an agent. If the agent is configured to run as a service on Linux, macOS, or Windows, then it is not running within an interactive logged-on session. Without an interactive logged-on session, UI interaction and other limitations exist.

## Get logs to diagnose problems

### Build logs

Start by looking at the logs in your completed build. If they don't provide enough detail, you can make them more verbose:

1. On the **Variables** tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. Queue the build.
3. In the explorer tab, view your completed build and click the build step to view its output.
4. If you need a copy of all the logs, click **Download all logs as zip**.

### Diagnostic logs

1. Log on to the agent machine.
2. Go to the `_diag` subfolder in the directory where the build agent is installed. For example:  
`c:\agent\_diag`

### 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 Team Foundation Server or VSTS account.
  - Shows when each job was run, and how it completed

Both logs show how the agent capabilities were detected and set.

## HTTP trace logs

**Important:** HTTP traces and trace files can contain passwords and other secrets. Do **not** post them on a public sites.

### Use built-in HTTP tracing

If your agent is version 2.114.0 or newer, you can trace the HTTP traffic headers and write them into the diagnostic log. Set the `VSTS_AGENT_HTTPTRACE` environment variable before you launch the `agent.listener`.

```
Windows:  
set VSTS_AGENT_HTTPTRACE=true  
  
macOS/Linux:  
export VSTS_AGENT_HTTPTRACE=true
```

### Use full HTTP tracing

#### Windows

1. Start [Fiddler](#).
2. We recommend you listen only to agent traffic. File > Capture Traffic off (F12)
3. Enable decrypting HTTPS traffic. Tools > Fiddler Options > HTTPS tab. Decrypt HTTPS traffic
4. Let the agent know to use the proxy:

```
set VSTS_HTTP_PROXY=http://127.0.0.1:8888
```

5. Run the agent interactively. If you're running as a service, you can set as the environment variable in control panel for the account the service is running as.

6. Restart the agent.

#### macOS and Linux

Use Charles Proxy (similar to Fiddler on Windows) to capture the HTTP trace of the agent.

1. Start Charles Proxy.
2. Charles: Proxy > Proxy Settings > SSL Tab. Enable. Add URL.
3. Charles: Proxy > Mac OSX Proxy. Recommend disabling to only see agent traffic.

```
export VSTS_HTTP_PROXY=http://127.0.0.1:8888
```

4. Run the agent interactively. If it's running as a service, you can set in the .env file. See [nix service](#)

5. Restart the agent.

## File- and folder-in-use errors

File or folder in use errors are often indicated by error messages such as:

Access to the path [...] is denied.

The process cannot access the file [...] because it is being used by another process.

Access is denied.

Can't move [...] to [...]

### Detect files and folders in use

On Windows, tools like [Process Monitor](#) can be used to capture a trace of file events under a specific directory.

Or, for a snapshot in time, tools like [Process Explorer](#) or [Handle](#) can be used.

### Anti-virus exclusion

Anti-virus software scanning your files can cause file or folder in use errors during a build. 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 steps 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 steps to force MSBuild to run only one process at a time.

File-in-use issues may result when leveraging the concurrent-process feature of MSBuild. Not specifying the argument `/maxcpucount:[n]` (short form `/m:[n]`) instructs MSBuild to use a single process only. If you are using the MSBuild or Visual Studio Build tasks, you may need to specify `/m:1` to override the `/m` argument that is added by default.

## Intermittent or inconsistent MSBuild failures

If you are experiencing intermittent or inconsistent MSBuild failures, try instructing MSBuild to use a single-process only. Intermittent or inconsistent errors may indicate that your target configuration is incompatible with the concurrent-process feature of MSBuild. See [MSBuild and /maxcpucount:\[n\]](#)

## Process hang

## Waiting for Input

A process hang may indicate that a process is waiting for input.

Running the agent from the command line of an interactive logged on session may help to identify whether a process is prompting with a dialog for input.

Running the agent as a service may help to eliminate programs from prompting for input. For example in .Net, programs may rely on the System.Environment.UserInteractive Boolean to determine whether to prompt. When running as a Windows service, the value is false.

## Process dump

Analyzing a dump of the process can help to identify what a deadlocked process is waiting on.

## WiX project

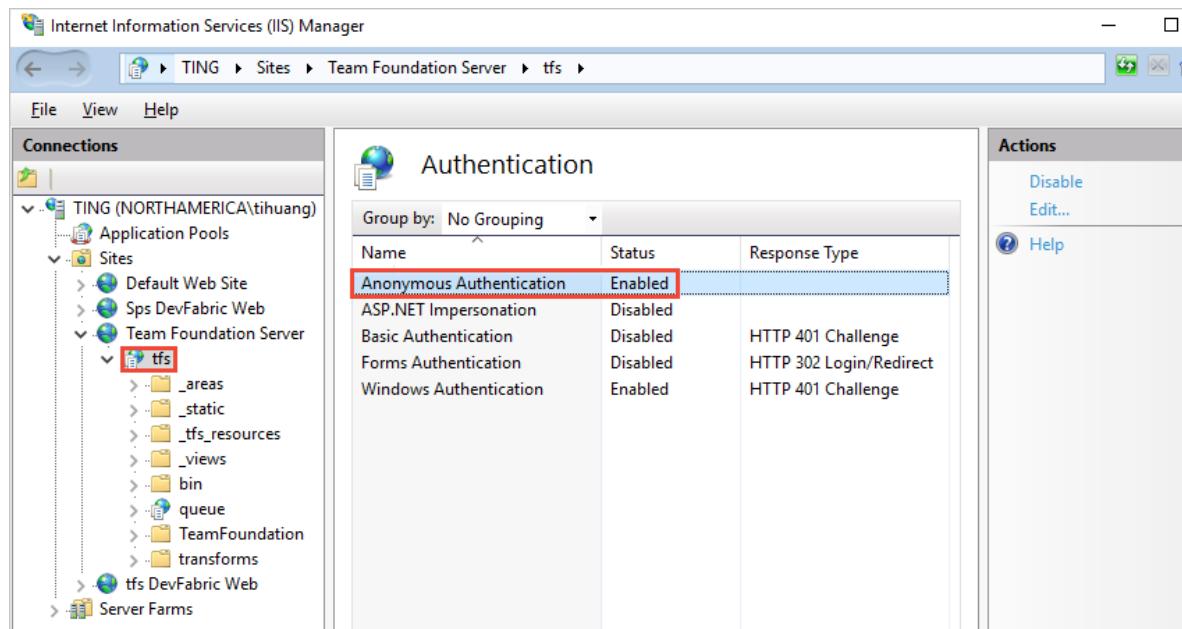
Building a WiX project when custom MSBuild loggers are enabled, can cause WiX to deadlock waiting on the output stream. Adding the additional MSBuild argument `/p:RunWixToolsOutOfProc=true` will workaround the issue.

# Agent connection issues

## Config fails while testing agent connection (on-premises TFS only)

```
Testing agent connection.  
VS30063: You are not authorized to access http://<SERVER>:8080/tfs
```

If the above error is received while configuring the agent, log on to your TFS machine. Start the Internet Information Services (IIS) manager. Make sure **Anonymous Authentication** is enabled.



## 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 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 VSTS, then you would want to ping an internet domain, for example bing.com. If you are using an on-premises TFS server, then you would want to ping a server on the same network.
- Verify the network throughput of the machine is adequate. You can perform an online speed test to check the throughput.
- If you use a proxy, verify the agent is configured to use your proxy. Refer to the agent deployment topic.

## Builds not starting

### TFS Job Agent not started

This may be characterized by a message in the web console "Waiting for an agent to be requested". Verify the TFSJobAgent (display name: *Visual Studio Team Foundation Background Job Agent*) Windows service is started.

### Misconfigured notification URL (1.x agent version)

This may be characterized by a message in the web console "Waiting for console output from an agent", and the build eventually times out.

A mismatching notification URL may cause the worker to process to fail to connect to the server. See *Team Foundation Administration Console, Application Tier*. The 1.x agent listens to the message queue using the URL that it was configured with. However, when a job message is pulled from the queue, the worker process uses the notification URL to communicate back to the server.

## Team Foundation Version Control (TFVC)

### Get sources not downloading some files

This may be characterized by a message in the build log "All files up to date" from the `tf get` command. Verify the built-in build 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 definition. In the version control web UI, you can browse the project files at any level of the folder hierarchy and check the security settings.

### Get sources through Team Foundation Proxy

The easiest way to configure the agent to get sources through a Team Foundation Proxy is set environment variable `TFSPROXY` that point to the TFVC proxy server for the agent's run as user.

```
Windows:  
set TFSPROXY=http://tfvcproxy:8081  
setx TFSPROXY=http://tfvcproxy:8081 // If the agent service is running as NETWORKSERVICE or any  
service account you can't easily set user level environment variable  
macOS/Linux:  
export TFSPROXY=http://tfvcproxy:8081
```

I need more help. I found a bug. I've got a suggestion. Where do I go?

[Get subscription, billing, and technical support](#)

Report any problems on [Developer Community](#).

We welcome your suggestions:

- Propose and vote on ideas on [UserVoice](#).

# Variables in Release Management

2/26/2018 • 11 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

As you compose the tasks for deploying your application into each environment, variables will help you to:

- Define a more generic deployment process once, and then customize it easily for each environment. 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 environment to another. These are **custom variables**.
- Use information about the context of the particular release, [environment](#), [artifacts](#), or [agent](#) in which the deployment process is being run. For example, your script may need access to the location of the build to download it, or to the working directory on the agent to create temporary files. These are **default variables**.

You can also use a default variable to [run a release in debug mode](#).

## Custom variables

Custom variables can be defined at various scopes.

- Share values across all of the definitions in a project by using [variable groups](#). Choose a variable group when you need to use the same values across all the definitions, environments, 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 environments by using [release definition variables](#). Choose a release definition variable when you need to use the same value across all the environments and tasks in the release definition, 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 definition.
- Share values across all of the tasks within one specific environment by using [environment variables](#). Use an environment-level variable for values that vary from environment to environment (and are the same for all the tasks in an environment). You define and manage these variables in the [Variables](#) tab of an environment in a release definition.

Using custom variables at project, release definition, and environment 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 definitions. Designate a configuration property to be a secure (secret) variable by selecting the  (padlock) icon next to the variable.

The values of hidden (secret) variables are stored securely on the server and cannot be viewed by users after they are saved. During a deployment, the Release Management service decrypts these values when referenced by the tasks and passes them to the agent over a secure HTTPS channel.

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)`.

You can use custom variables to prompt for values during the execution of a release. For more details, see [Approvals](#).

## Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command.

### 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 `secretSauce` variables

```
@echo ##vso[task.setvariable variable=sauce]crushed tomatoes
@echo ##vso[task.setvariable variable=secretSauce;issecret=true]crushed tomatoes with garlic
```

 Read the variables

### Arguments

```
"$(sauce)" "$(secretSauce)"
```

### Script

```
@echo off
set sauceArgument=%~1
set secretSauceArgument=%~2
@echo No problem reading %sauceArgument% or %SAUCE%
@echo But I cannot read %SECRETSAUCE%
@echo But I can read %secretSauceArgument% (but the log is redacted so I do not spoil
the secret)
```

Console output from reading the variables:

```
No problem reading crushed tomatoes or crushed tomatoes
But I cannot read
But I can read ***** (but the log is redacted so I do not spoil the secret)
```

## Default variables

Information about the execution context is made available to running tasks through default variables. Your tasks and scripts can use these variables to find information about the system, release, environment, 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.

## System variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.TeamFoundationServerUri	The URL of the Release Management service endpoint in the TFS or VSTS account. Use this from your scripts or tasks to call REST APIs on the Release Management service.	https://fabrikam.vssrm.visualstudio.com/	
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or VSTS account. Use this from your scripts or tasks to call REST APIs on other services such as Build and Version control.	https://fabrikam.visualstudio.com/	
System.CollectionId	The ID of the collection to which this build or release belongs.	6c6f3423-1c84-4625-995a-f7f143a1e43d	TFS 2015
System.TeamProject	The name of the team project to which this build or release belongs.	Fabrikam	
System.TeamProjectId	The ID of the team project to which this build or release belongs.	79f5c12e-3337-4151-be41-a268d2c73344	TFS 2015
System.ArtifactsDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
System.DefaultWorkingDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.ArtifactsDirectory.	C:\agent_work\r1\a	
System.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.	C:\agent_work	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.Debug	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.	true	

## Release variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.DefinitionName	The name of the release definition to which the current release belongs.	fabrikam-cd	
Release.DefinitionId	The ID of the release definition to which the current release belongs.	1	TFS 2015
Release.ReleaseName	The name of the current release.	Release-47	
Release.ReleaseId	The identifier of the current release record.	118	
Release.ReleaseUri	The URI of current release.	vstfs:///ReleaseManagement/Release/118	
Release.ReleaseDescription	The text description provided at the time of the release.	Critical security patch	
Release.RequestedFor	The display name of identity that triggered the release.	Mateo Escobedo	
Release.RequestedForId	The ID of identity that triggered the release.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	
Release.EnvironmentName	The name of environment to which deployment is currently in progress.	Dev	
Release.EnvironmentId	The ID of the environment instance in a release to which the deployment is currently in progress.	276	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.EnvironmentUri	The URI of environment instance in a release to which deployment is currently in progress.	vstfs://ReleaseManagement/Environment/276	
Release.DefinitionEnvironmentId	The ID of the environment in the corresponding release definition.	1	TFS 2015
Release.AttemptNumber	The number of times this release is deployed in this environment.	1	TFS 2015
Release.Deployment.RequestedFor	The display name of the identity that triggered (started) the deployment currently in progress.	Mateo Escobedo	TFS 2015
Release.Deployment.RequestedForId	The ID of the identity that triggered (started) the deployment currently in progress.	2f435d07-769f-4e46-849d-10d1ab9ba6ab	TFS 2015

### Release environment variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.Environments.{Environment name}.Status	The status of deployment of this release within a specified environment.	NotStarted	TFS 2015

### Agent variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
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.	fabrikam-agent	
Agent.MachineName	The name of the computer on which the agent is configured.	fabrikam-agent	
Agent.Version	The version of the agent software.	2.109.1	
Agent.JobName	The name of the job that is running, such as Release or Build.	Release	
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.	C:\agent	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.ReleaseDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as System.ArtifactsDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
Agent.RootDirectory	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.WorkFolder and System.WorkFolder.	C:\agent_work	
Agent.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and System.WorkFolder.	C:\agent_work	
Agent.DeploymentGroupId	The ID of the deployment group the agent is registered with. This is available only in deployment group phases.	1	TFS 2018 U1

## Artifact variables

For each artifact that is referenced in a release, you can use the following artifact variables. Not all variables are meaningful for each artifact type. The table below lists the default artifact variables and provides examples of the values that they have depending on the artifact type. If an example is empty, it implies that the variable is not populated for that artifact type.

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{Artifact alias}.DefinitionId	The identifier of the build definition or repository.	1			fabrikam/asp
Release.Artifacts.{Artifact alias}.DefinitionName	The name of the build definition or repository.	fabrikam-ci		TFVC: \$/fabrikam, Git: fabrikam	fabrikam/asp (master)
Release.Artifacts.{Artifact alias}.BuildNumber	The build number or the commit identifier.	20170112.1	20170112.1	TFVC: Changeset 3, Git: 38629c964	38629c964

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/ TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{Artifact alias}.BuildId	The build identifier.	130	130		38629c964d21fe 405ef830b7d022 0966b82c9e11
Release.Artifacts.{Artifact alias}.BuildURI	The URL for the build.	vstfs:///build-release /Build/130			
Release.Artifacts.{Artifact alias}.SourceBranch	The path of the branch from which the source was built.	refs/heads/master			
Release.Artifacts.{Artifact alias}.SourceBranchName	The name of the branch from which the source was built.	master			
Release.Artifacts.{Artifact alias}.SourceVersion	The commit that was built.	bc0044458ba1d9298 cdc649cb5dcf01 3180706f7			
Release.Artifacts.{Artifact alias}.Repository.Provider	The type of repository from which the source was built	Git			
Release.Artifacts.{Artifact alias}.RequestedForID	The identifier of the account that triggered the build.	2f435d07-769f-4e46-849d-10d1ab9ba6ab			
Release.Artifacts.{Artifact alias}.RequestedFor	The name of the account that requested the build.	Mateo Escobedo			
Release.Artifacts.{Artifact alias}.Type	The type of artifact source, such as Build.	Build	Jenkins: Jenkins, TeamCity: TeamCity	TFVC: TFVC, Git: Git	GitHub

See also [Artifact source alias](#)

### Primary artifact variables

You designate one of the artifacts as a primary artifact in a release definition. For the designated primary artifact, Release Management populates the following variables.

VARIABLE NAME	SAME AS
Build.DefinitionId	Release.Artifacts.{Primary artifact alias}.DefinitionId
Build.DefinitionName	Release.Artifacts.{Primary artifact alias}.DefinitionName

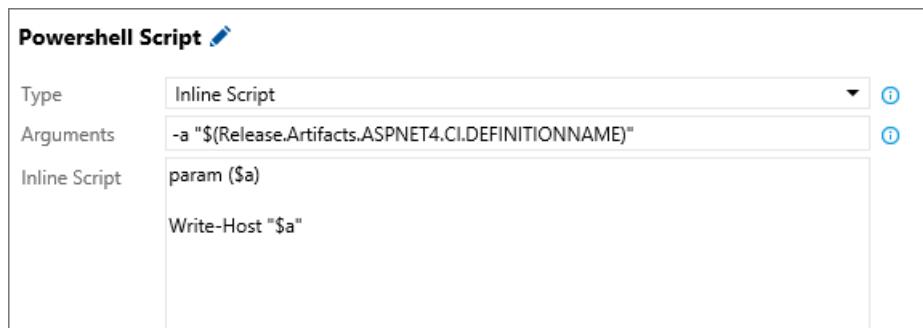
VARIABLE NAME	SAME AS
Build.BuildNumber	Release.Artifacts.{Primary artifact alias}.BuildNumber
Build.BuildID	Release.Artifacts.{Primary artifact alias}.BuildId
Build.BuildURI	Release.Artifacts.{Primary artifact alias}.BuildURI
Build.SourceBranch	Release.Artifacts.{Primary artifact alias}.SourceBranch
Build.SourceBranchName	Release.Artifacts.{Primary artifact alias}.SourceBranchName
Build.SourceVersion	Release.Artifacts.{Primary artifact alias}.SourceVersion
Build.Repository.Provider	Release.Artifacts.{Primary artifact alias}.Repository.Provider
Build.RequestedForID	Release.Artifacts.{Primary artifact alias}.RequestedForID
Build.RequestedFor	Release.Artifacts.{Primary artifact alias}.RequestedFor
Build.Type	Release.Artifacts.{Primary artifact alias}.Type

## Using default variables

You can use the default variables in two ways - as parameters to tasks in a release definition or in your scripts.

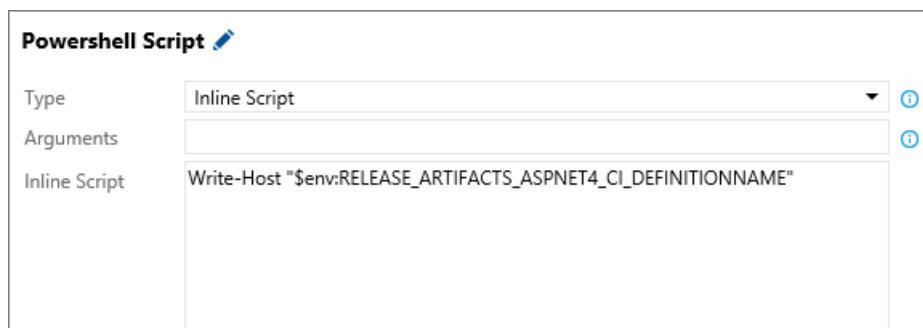
You can directly use a default variable as an input to a task. For example, to pass

`Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** to a task, you would use `$(Release.Artifacts.ASPNET4.CI.DefinitionName)`.



To use a default variable in your script, you must first replace the `.` in the default variable names with `_`. For example, to print the value of artifact variable `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** in a Powershell script, you would use

```
$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME .
```



Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

## 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 environment, 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 definition.
- To initiate debug mode for a single environment, open the **Configure environment** dialog from the shortcut menu of the environment 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 definition.

If you get an error related to an Azure RM service endpoint, see [How to: Troubleshoot Azure Resource Manager service endpoints](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# How to: Troubleshoot Azure Resource Manager service endpoints

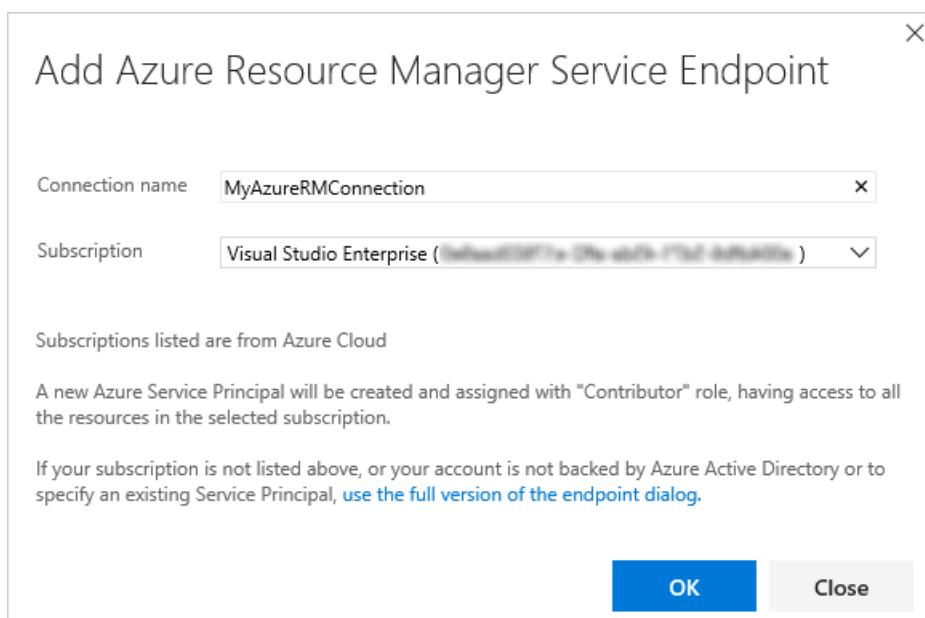
2/26/2018 • 3 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

This topic will help you resolve issues you may encounter when creating a connection to Microsoft Azure using an **Azure Resource Manager** service endpoint.

## What happens when you create a Resource Manager service endpoint?

You open the **Add Azure Resource Manager Service Endpoint** dialog, provide a connection name, and select a subscription from drop-down list of your subscriptions.



When you choose **OK**, the system:

1. Connects to the Azure Active Directory (AAD) tenant for the selected subscription
2. Creates an application in AAD 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 endpoint using this application's details

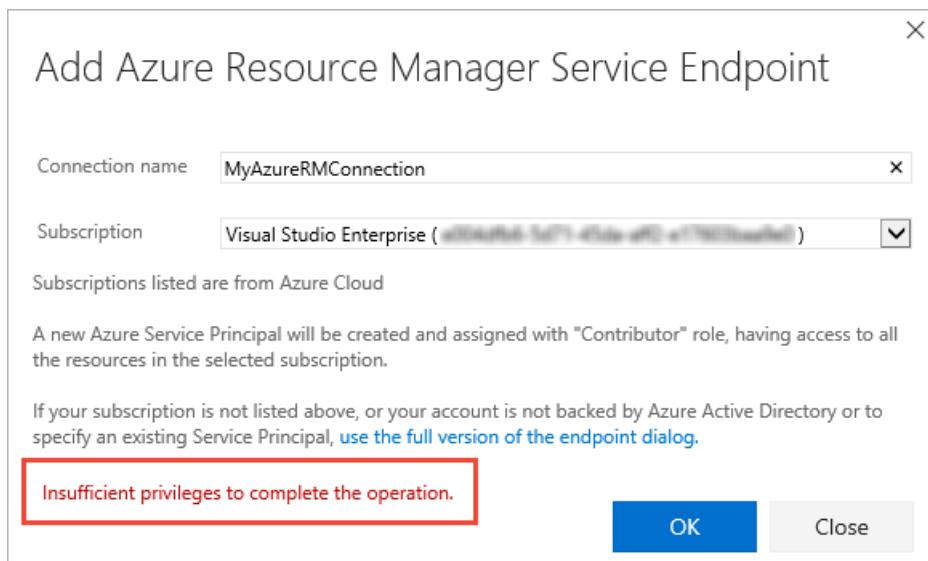
## How to troubleshoot errors that may occur

Errors that may occur when the system attempts to create the service endpoint include:

- [Insufficient privileges to complete the operation](#)
- [Failed to obtain an access token](#)
- [A valid refresh token was not found](#)
- [Failed to assign contributor role](#)

### **Insufficient privileges to complete the operation**

This typically occurs when the system attempts to create an application in AAD 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 into the Azure portal at <https://portal.azure.com> using an Administrator account.
2. Choose **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 choose **Users**.
5. Choose **User settings**.
6. In the **External users** section, change **Guest user permissions are limited to No**.

Alternatively, if you are prepared to give the user additional (administrator-level) permissions, you can make the user a member of the **Global administrator** role as follows:

1. Sign into the Azure portal at <https://portal.azure.com> using an Administrator account.
2. Choose **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 choose **Users**.
5. Use the search box to filter the list and then choose the user you want to manage.
6. In the **MANAGE** section choose **Directory role** and change the role to **Global administrator**.
7. Save the change.

It typically takes 15 to 20 minutes to apply the changes globally. After this period has elapsed, the user can retry creating the service endpoint.

#### **The user is not authorized to add applications in the directory**

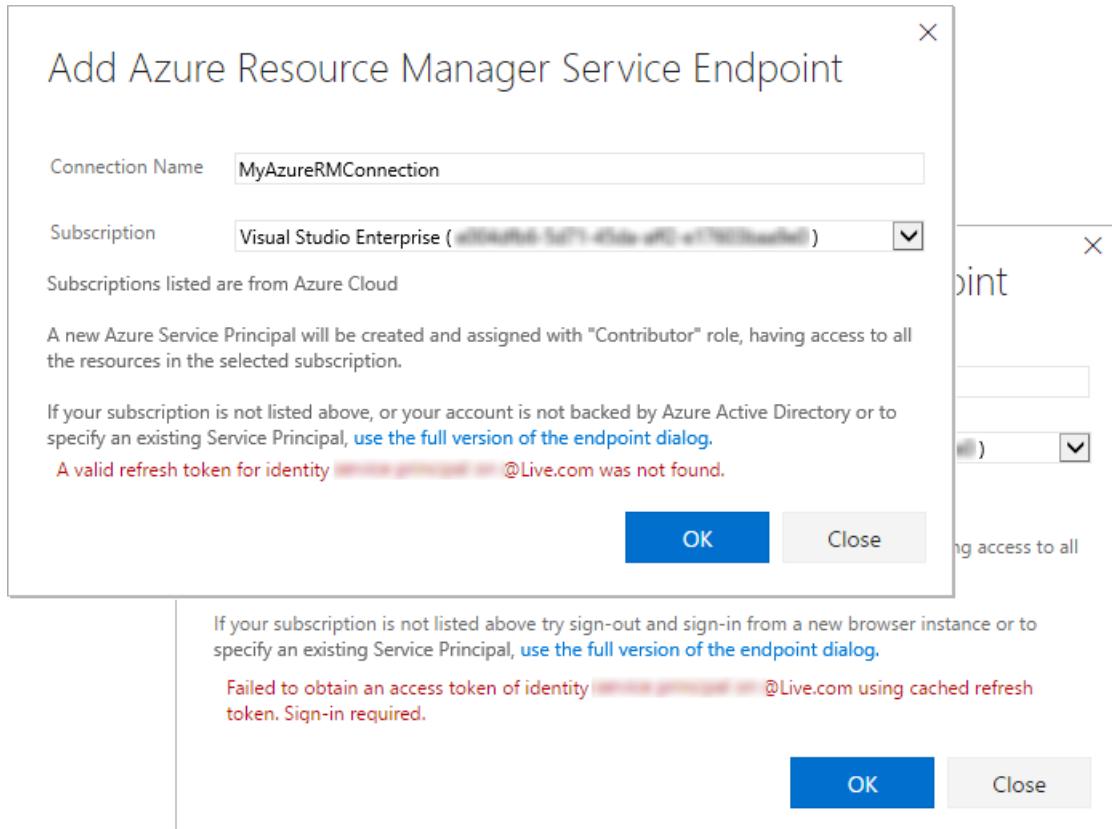
You must have permission to add integrated applications in the directory. The directory administrator has

permission to change this setting, as follows:

1. Choose **Azure Active Directory** in the left navigation bar.
2. Ensure you are editing the appropriate directory corresponding to the user subscription. If not, select **Switch directory** and log in using the appropriate credentials if required.
3. In the **MANAGE** section choose **Users**.
4. Choose **User settings**.
5. In the **App registrations** section, change **Users can register applications** to **Yes**.

#### Failed to obtain an access token or A valid refresh token was not found

These errors typically occur when your session has expired.

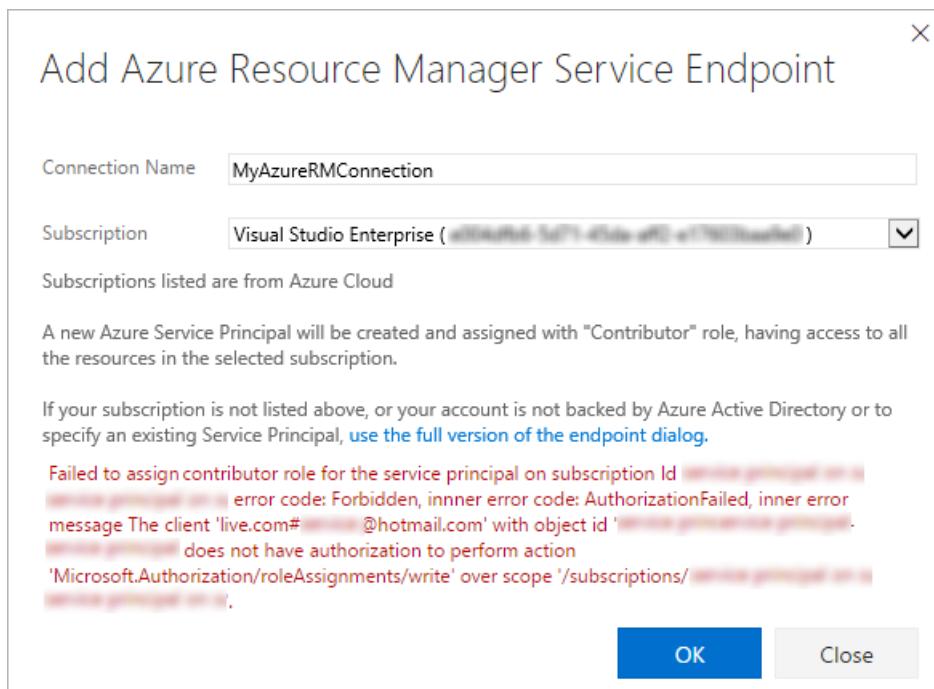


To resolve these issues:

- Sign out of VSTS or TFS.
- Open an InPrivate or incognito browser window and navigate to <https://www.visualstudio.com/team-services/>.
- If you are prompted to sign out, do so.
- Sign in using the appropriate credentials.
- Choose the account you want to use from the list.
- Select the project you want to add the service endpoint to.
- Create the service endpoint you need by opening the **Settings** page, selecting the **Services** tab, choosing **New service endpoint**, and selecting **Azure Resource Manager**.

#### Failed to assign Contributor role

This error typically occurs when you do not have **Write** permission for the selected Azure subscription when the system attempts to assign the **Contributor** role.



To resolve this issue, ask the subscription administrator to configure an **Admin Access** role for your identity.

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Build and release tasks

2/21/2018 • 7 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)**

## Build

TASK	VERSIONS
 Android Build - deprecated. Use <a href="#">Gradle</a>	VSTS, TFS 2015 RTM and newer
 <a href="#">Android Signing</a> . Sign and align Android APK files	VSTS, TFS 2015 RTM and newer
 <a href="#">Ant</a> . Build with Apache Ant	VSTS, TFS 2015 RTM and newer
 <a href="#">CMake</a> . Build with the CMake cross-platform build system	VSTS, TFS 2015 RTM and newer
 <a href="#">Gradle</a> . Build using a Gradle wrapper script	VSTS, TFS 2015 RTM and newer
 <a href="#">Grunt</a> . The JavaScript Task Runner	VSTS, TFS 2015.3 and newer
 <a href="#">Gulp</a> . Node.js streaming task based build system	VSTS, TFS 2015 RTM and newer
 <a href="#">Index Sources &amp; Publish Symbols</a> . Index your source code and publish symbols to a file share	VSTS, TFS 2015 RTM and newer
 <a href="#">Jenkins Queue Job</a> . Queue a job on a Jenkins server	VSTS, TFS 2015 RTM and newer
 <a href="#">Maven</a> . Build with Apache Maven	VSTS, TFS 2015 RTM and newer
 <a href="#">MSBuild</a> . Build with MSBuild	VSTS, TFS 2015 RTM and newer
 <a href="#">.NET Core CLI</a> . Build, test, and publish using the .NET Core command line	VSTS, TFS 2018
 <a href="#">Publish Build Artifacts</a> . Publish Build artifacts to the server or a file share	TFS 2015 RTM. Deprecated on VSTS and newer versions of TFS.
<a href="#">SonarQube for MSBuild - Begin Analysis</a> . Fetch the Quality Profile from SonarQube to configure the analysis	VSTS, TFS 2015.3 and newer

TASK	VERSIONS
SonarQube for MSBuild - End Analysis. Finish the analysis and upload the results to SonarQube	VSTS, TFS 2015.3 and newer
 Visual Studio Build Visual Studio version property	. Build with MSBuild and set the Visual Studio version property VSTS, TFS 2015 RTM and newer
 Xamarin.Android	. Build an Android app with Xamarin VSTS, TFS 2015 RTM and newer
 Xamarin.iOS	. Build an iOS app with Xamarin on macOS VSTS, TFS 2015 RTM and newer
 Xcode	. Build an Xcode workspace on macOS VSTS, TFS 2015 RTM and newer
 Xcode Package iOS build output	. Generate an .ipa file from Xcode build output VSTS, TFS 2015 RTM and newer

## Utility

TASK	VERSIONS
 Archive Files	. Archive files using a variety of compression formats such as .7z, .rar, .tar.gz, and .zip. VSTS, TFS 2017 and newer
 Azure function	. Invoke a HTTP triggered function in an Azure function app and parse the response. VSTS
 Azure monitor	. Observe the configured Azure monitor rules for active alerts. VSTS
 Batch Script	. Run a windows cmd or bat script and optionally allow it to change the environment VSTS, TFS 2015 RTM and newer
 Command Line	. Run a command line with arguments VSTS, TFS 2015 RTM and newer
 Copy and Publish Build Artifacts	. Copy Build artifacts to staging folder then publish Build artifacts to the server or a file share TFS 2015 RTM. Deprecated on VSTS and newer versions of TFS.
 Copy Files	. Copy files from source folder to target folder using minimatch patterns (The minimatch patterns will only match file paths, not folder paths). VSTS, TFS 2015.3 and newer

TASK	VERSIONS
 <a href="#">cURL Upload Files</a> . Use cURL to upload files with supported protocols. (FTP, FTPS, SFTP, HTTP, and more)	VSTS, TFS 2015 RTM and newer
 <a href="#">Delay</a> . Pause execution of the process for a fixed delay time.	VSTS
 <a href="#">Delete Files</a> . Delete files or folders.	VSTS, TFS 2015.3 and newer
 <a href="#">Download Secure File</a> . Download a secure file to a temporary location on the build or release agent.	VSTS
 <a href="#">Extract Files</a> . Extract files from archives (.zip, .jar, .war, .ear, .tar, .7z, and others) to a target folder.	VSTS, TFS 2017 and newer
 <a href="#">FTP Upload</a> . Upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS.	VSTS, TFS 2017 and newer
 <a href="#">Install Apple Certificate</a> required to build on a macOS agent.	VSTS, TFS 2018
 <a href="#">Install Apple Provisioning Profile</a> provisioning profile required to build on a macOS agent.	VSTS, TFS 2018
 <a href="#">Invoke HTTP REST API</a> the response.	VSTS
 <a href="#">Manual intervention</a> . Pause an active deployment within an environment, typically to perform some manual steps or actions, and then continue the automated deployment steps.	VSTS
 <a href="#">PowerShell</a> . Run a PowerShell script	VSTS, TFS 2015 RTM and newer
 <a href="#">Publish Build Artifacts</a> server or a file share	VSTS, TFS 2015.3 and newer
 <a href="#">Publish To Azure Service Bus</a> Azure Service Bus using a service connection and without using an agent.	VSTS
 <a href="#">Query Work Items</a> Ensure the number of matching items returned by a work item query is within the configured thresholds.	VSTS

TASK	VERSIONS
 <a href="#">Service Fabric PowerShell</a> Runs any PowerShell command or script in a PowerShell session that has a Service Fabric cluster connection initialized.	VSTS
 <a href="#">Shell Script</a> Run a shell script using bash	VSTS, TFS 2015 RTM and newer
 <a href="#">Update Service Fabric App Versions</a> Automatically updates the versions of a packaged Service Fabric application	VSTS, TFS 2017 and newer
 <a href="#">Xamarin License</a> Activate or deactivate Xamarin licenses	VSTS, TFS 2015 RTM and newer

## Test

TASK	VERSIONS
 <a href="#">Cloud-based Apache JMeter Load Test</a> Runs the Apache JMeter load test in cloud	VSTS, TFS 2015 RTM and newer
 <a href="#">Cloud-based Load Test</a> Runs the load test in cloud, with VSTS	VSTS, TFS 2015 RTM and newer
 <a href="#">Cloud-based Web Performance Test</a> Runs the quick web performance test in cloud, with VSTS	VSTS, TFS 2015 RTM and newer
 <a href="#">App Center Test</a> Test mobile app packages with Visual Studio App Center	VSTS, TFS 2015.3 and newer
 <a href="#">Publish Code Coverage Results</a> Publish code coverage results to VSTS/TFS	VSTS, TFS 2015.3 and newer
 <a href="#">Publish Test Results</a> Publish Test Results to VSTS/TFS	VSTS, TFS 2015 RTM and newer
 <a href="#">Visual Studio Test version 1</a> Run tests with Visual Studio test runner	VSTS, TFS 2015 RTM and newer
 <a href="#">Xamarin Test Cloud</a> Test mobile apps with Xamarin Test Cloud using Xamarin.UITest	VSTS, TFS 2015 RTM and newer

## Package

TASK	VERSIONS
 <a href="#">CocoaPods</a> for Swift and Objective-C Cocoa projects. Runs pod install	VSTS, TFS 2015 RTM and newer
 . Install npm packages	VSTS, TFS 2015 RTM and newer
 <a href="#">NuGet Installer</a> packages	VSTS, TFS 2015 RTM and newer
 <a href="#">NuGet Packager</a> Creates nupkg outputs from csproj or nuspec files	VSTS, TFS 2015.3 and newer
 <a href="#">NuGet Publisher</a> server	VSTS, TFS 2015.3 and newer
 <a href="#">Xamarin component restore</a> components for the specified solution	VSTS, TFS 2017 and newer

## Deploy

TASK	VERSIONS
 <a href="#">Azure App Service Deploy</a> using Web Deploy / Kudu REST APIs	VSTS, TFS 2017 and newer
 <a href="#">Azure App Service Manage</a> swap for an Azure App Service	VSTS, TFS 2017 and newer
 <a href="#">Azure CLI</a> . Run a shell or batch script containing Azure CLI commands against an Azure subscription	VSTS, TFS 2017 and newer
 <a href="#">Azure Cloud Service Deployment</a> . Deploy an Azure Cloud Service	VSTS, TFS 2015 RTM and newer
 <a href="#">Azure File Copy</a>	VSTS, TFS 2015.3 and newer
 <a href="#">Azure Key Vault</a> Key Vault into a release definition	VSTS
 <a href="#">Azure PowerShell</a> . Run a PowerShell script within an Azure environment	VSTS, TFS 2015 RTM and newer
 <a href="#">Azure Resource Group Deployment</a> delete Azure Resource Groups	VSTS, TFS 2015.3 and newer

TASK	VERSIONS
 <a href="#">Azure SQL Database Deployment</a> DB using DACPAC	VSTS, TFS 2015.3 and newer
 <a href="#">Copy Files Over SSH</a> . Copy files from source folder to target folder on a remote machine over SSH	VSTS, TFS 2017 and newer
 <a href="#">IIS Web App Deploy</a> . Deploy IIS Websites and Virtual Applications using WebDeploy	VSTS
 <a href="#">IIS Web App Manage</a> and recycle IIS Websites, IIS Web Applications, Virtual Directories, and IIS Application Pools	VSTS
 <a href="#">App Center Distribute</a> . Upload and distribute mobile app packages using Visual Studio App Center	VSTS, TFS 2015 RTM and newer
 <a href="#">PowerShell on Target Machines</a> . Execute PowerShell scripts on remote machine(s)	VSTS, TFS 2015 RTM and newer
 <a href="#">Service Fabric Application Deployment</a> . Deploy a Service Fabric application to a cluster	VSTS, TFS 2017 and newer
 <a href="#">Service Fabric Compose Deploy</a> . Deploy a Service Fabric application to a cluster using a compose file	VSTS
 <a href="#">SSH</a> . Run shell commands or a script on a remote machine using SSH	VSTS, TFS 2017 and newer
 <a href="#">Windows Machine File Copy</a> . Copy files to remote machine(s)	VSTS, TFS 2015 RTM and newer

## Tool

TASK	VERSIONS
 <a href="#">.NET Core Tool Installer</a> . Acquires a specific version of .NET Core and adds it to the PATH. Use the task to change the Core version for subsequent tasks.	VSTS, TFS 2018
 <a href="#">Node Tool Installer</a> . Finds or downloads and caches the specified version of Node.js and adds it to the PATH	VSTS

TASK	VERSIONS
 <b>Java Tool Installer</b> . Acquires a specific version of Java from a user supplied Azure blob, a location in the source or on the agent, or the tools cache and sets JAVA_HOME. Use this task to change the version of Java used in Java tasks.	VSTS

To learn more about tool installer tasks, see [Tool installers](#).

## Q&A

**Where can I learn step-by-step how to build my app?**

[Build your app](#)

**Can I add my own build tasks?**

Yes: [Add a build task](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: .NET Core

1/9/2018 • 6 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017



Build, test, and release .NET Core and .NET Standard projects and create .NET Core and .NET Standard NuGet packages using the `dotnet` command-line tool.

If your .NET Core or .NET Standard build depends on NuGet packages, make sure to add two copies of this step: one with the `restore` command and one with the `build` command.

## Restore NuGet packages

### Demand

None

### Arguments

ARGUMENT	DESCRIPTION
Path to project(s)	Copy the <b>Project(s)</b> argument in your <code>build</code> command step and paste it here, or create a link using the Link button in the information panel.

### FEEDS AND AUTHENTICATION

Feeds to use	<p><b>Feed(s) I select here:</b></p> <ul style="list-style-type: none"><li>Select this option to use NuGet.org and/or one Package Management feed in the same account/collection as the build.</li></ul> <p><b>Feeds in my NuGet.config:</b></p> <ul style="list-style-type: none"><li>Select this option to use feeds specified in a <a href="#">NuGet.config</a> file you've checked into source control.</li><li>Credentials for feeds outside this account/collection can be used to inject credentials you've provided as a <a href="#">NuGet service endpoint</a> into your NuGet.config as the build runs.</li></ul>
--------------	---

### ADVANCED

Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.

### CONTROL OPTIONS

# Pack NuGet packages

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>**\*.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none"><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 <a href="#">.nuspec file</a>. 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 <a href="#">.nuspec file</a> 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 <a href="#">.nuspec file</a>.</li></ul> <p>To package a single file, click the ... button and select the file. To package multiple files, use <a href="#">file matching patterns</a>. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a <a href="#">variable</a> such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	<p>This <a href="#">blog post</a> provides an overview of the automatic package versioning available here.</p>
ADVANCED	
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 <code>nuget pack</code> with the <code>-Properties</code> option.

ARGUMENT	DESCRIPTION
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

## Push NuGet packages

### Demands

None

### Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none"> <li>Default value: <code>\$(Build.ArtifactStagingDirectory)/*.nupkg</code></li> <li>To publish a single package, click the ... button and select the file.</li> <li>Use single-folder wildcards (<code>*</code>) and recursive wildcards (<code>**</code>) to publish multiple packages.</li> <li>Use <a href="#">variables</a> to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the <b>package folder</b> in the pack step above, you could specify <code>\$(Build.ArtifactStagingDirectory)\**\*.nupkg</code> here.</li> </ul>
Target feed location	<ul style="list-style-type: none"> <li><b>This account/collection</b> publishes to a Package Management feed in the same account/collection as the build. After you select this option, select the target feed from the dropdown.           <ul style="list-style-type: none"> <li>"Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. This option is currently only available on VSTS.</li> </ul> </li> <li><b>External NuGet server (including other accounts/collections)</b> publishes to an external server such as <a href="#">NuGet</a>, <a href="#">MyGet</a>, or a Package Management feed in another VSTS account or TFS collection. After you select this option, you create and select a <a href="#">NuGet service endpoint</a>.</li> </ul>
ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

## Custom NuGet command

### Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

## Q & A

### Why is my build or publish step failing to restore packages?

Most `dotnet` commands, including `build` and `publish`, include an implicit `restore` step. This will fail against authenticated feeds, even if you ran a successful `dotnet restore` in an earlier step, because the earlier step will have cleaned up the credentials it used.

To fix this issue, add the `--no-restore` flag to the Arguments textbox.

### Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add a VSTS 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 VSTS package management?

[Package Management in VSTS and TFS](#)

### Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

### What other kinds of apps can I build?

[Build your app](#)

### What other kinds of build steps are available?

[Specify your build steps](#)

### 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. This option is not available for GitHub repos.
- TFVC: [Use gated check-in](#).

### How do I modify other parts of my build definition?

- [Specify your build steps](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

**I selected parallel multi-configuration, but only one build is running at a time.**

If you're using VSTS, you might need more concurrent pipelines. See [Concurrent build and release pipelines in VSTS](#).

**How do I see what has changed in my build definition?**

[View the change history of your build definition](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

4 min to read •

# Build: Android build (deprecated; use Gradle)

11/14/2017 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



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	The location in the repository of the gradlew wrapper used for the build. For agents on Windows (including 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> .
Project Directory	Relative path from the repo root to the root directory of the application (likely where your build.gradle file is).
Gradle Arguments	Provide any options to pass to the Gradle command line. The default value is <code>build</code>  See <a href="#">Gradle command line</a> .
ANDROID VIRTUAL DEVICE (AVD) OPTIONS	
Name	Name of the AVD to be started or created.  <b>Note:</b> You must deploy your own <a href="#">agent</a> to use this option. You cannot use a hosted pool if you want to create an AVD.
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 definition 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 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 steps

[Android Signing](#)

# Build: Android signing

9/12/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Sign and align Android APK files

## Demands

The build agent must have the following capabilities:

- Java JDK

## Arguments

ARGUMENT	DESCRIPTION
APK Files	<p>Relative path from the repo root to the APK(s) you want to sign. You can use wildcards to specify multiple files. For example:</p> <ul style="list-style-type: none"><li><code>outputs\apk\*.apk</code> to sign all .APK files in the <code>outputs\apk\</code> subfolder</li><li><code>**\\bin\\*.apk</code> to sign all .APK files in all bin subfolders</li></ul>
<b>SIGNING OPTIONS</b>	
Sign the APK	Select this option to sign the APK with a provided keystore file. Unsigned APKs can only run in an emulator. APKs must be signed to run on a device.
Keystore File	Enter the file path to the keystore file that should be used to sign the APK. It can either be checked into source control or placed on the build machine directly by an administrator. It is recommended to encrypt the keystore file in source control and use the <b>Decrypt File</b> task to decrypt the file during the build.
Keystore Password	Enter the password for the provided keystore file. <b>Important:</b> We recommend that you put this value in a <a href="#">secret variable</a> .
Alias	Enter the alias that identifies the public/private key pair to be used in the keystore file.

ARGUMENT	DESCRIPTION
Key Password	<p>Enter the key password for the alias and keystore file.</p> <p><b>Important:</b> We recommend that you put this value in a <a href="#">secret variable</a>.</p>
Jarsigner Arguments	<p>Provide any options to pass to the jarsigner command line. Default is</p> <pre>-verbose -sigalg MD5withRSA -digestalg SHA1</pre> <p>See <a href="#">jarsigner documentation</a>.</p>
<b>ZIPALIGN OPTIONS</b>	
Zipalign	Select if you want to zipalign your package. This reduces the amount of RAM consumed by an app.
Zipalign Location	(Optional) The location of the zipalign executable used during signing. Defaults to the zipalign found in the Android SDK version folder your application builds against.
<b>CONTROL OPTIONS</b>	

## Related steps

[Android Build](#)

# Build: Ant

9/13/2017 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Build with Apache Ant

## Demands

The build agent must have the following capability:

- Apache Ant

## Arguments

ARGUMENT	DESCRIPTION
Ant Build File	Relative path from the repository root to the Ant build file. For more information about build files, see <a href="#">Using Apache Ant</a> .
Options	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> .
Target(s)	Target(s) for Ant to execute for this build. See <a href="#">Using Apache Ant Targets</a> .

## JUNIT TEST RESULTS

Publish to VSTS/TFS	Select this option to publish JUnit test results produced by the Ant build to VSTS or your on-premises Team Foundation Server. Each test result file that matches Test Results Files is published as a test run.
Test Results Files	Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all xml files whose name starts with TEST-.
Test Run Title	Assign a title for the JUnit test case results for this build.

## CODE COVERAGE

ARGUMENT	DESCRIPTION
Code Coverage Tool	<p>Select the code coverage tool you want to use.</p> <p>If you are using the <a href="#">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:</p> <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> <p>After you select one of these tools, the following arguments appear.</p>
Class Files Directories	<p>Specify a comma-separated list of relative paths from the Ant build file to the directories that contain your .class files, archive files (such as .jar and .war). Code coverage is reported for class files present in the directories. Directories and archives are searched recursively for class files. For example: target/classes,target/testClasses.</p>
Class Inclusion/Exclusion Filters	<p>Specify a comma-separated list of filters to include or exclude classes from collecting code coverage. For example: +:com.,+:org.,-:my.app.</p>
Source Files Directories	<p>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>
<b>ADVANCED</b>	
Set ANT_HOME Path	If set, overrides any existing ANT_HOME environment variable with the given path.
Set JAVA_HOME by JDK Version	Choose which JDK level will be used to run Ant. Will attempt to find JDK version and assign JAVA_HOME before running Ant.
Set JAVA_HOME by Path	Directory on build agent where the JDK is located.
JDK Architecture	Optionally supply the architecture (x86, x64) of the JDK.
<b>CONTROL OPTIONS</b>	

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

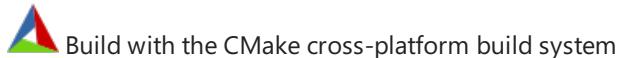
**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: CMake

11/14/2017 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



## Demands

cmake

## Arguments

ARGUMENT	DESCRIPTION
Working Directory	<p>Working directory when CMake is run. The default value is <code>build</code>.</p> <p>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>.</p> <p>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></p> <p>If the path you specify does not exist, CMAke creates it.</p>
Arguments	Arguments that you want to pass to CMake.
CONTROL OPTIONS	

## Q&A

### How do I enable CMake for hosted agents?

The [hosted agents](#) have CMake installed, but you must manually add the [capability](#) to use the CMake build step.

1. Open the Agent Pools control panel tab:

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{your\\_server}/tfs/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

2. In the left column, click the name of the hosted pool that you are using. In the right column click **Capabilities**.

3. Click **Add capability** and set the fields to `cmake` and `yes`.

4. Click **Save changes**

### How do I enable CMake for my on-premises agent?

1. [Deploy an agent](#).

2. [Install CMake](#) and make sure to add it to the path of the user that the agent is running as on your agent machine.

3. In your web browser, navigate to the **Agent pools** control panel tab:

- VSTS: [https://{your\\_account}.visualstudio.com/\\_admin/\\_AgentPool](https://{your_account}.visualstudio.com/_admin/_AgentPool)
- TFS 2017 and newer: [https://{your\\_server}/tfss/DefaultCollection/\\_admin/\\_AgentPool](https://{your_server}/tfss/DefaultCollection/_admin/_AgentPool)
- TFS 2015: [http://{your\\_server}:8080/tfs/\\_admin/\\_AgentPool](http://{your_server}:8080/tfs/_admin/_AgentPool)

[The TFS URL doesn't work for me. How can I get the correct URL?](#)

4. In the left column, click the name of your agent pool. In the right column click **Capabilities**.

5. Click **Add capability** and set the fields to `cmake` and `yes`.

6. Click **Save changes**

**How does CMake work? What arguments can I use?**

[About CMake](#)

[CMake Documentation](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Gradle

11/14/2017 • 3 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Build using a Gradle wrapper script

## Arguments

ARGUMENT	DESCRIPTION
Gradle Wrapper	<p>The location in the repository of the gradlew wrapper used for the build. For agents on Windows (including 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>
Options	<p>Specify any command line options you want to pass to the Gradle wrapper.</p> <p>See <a href="#">Gradle Command Line</a>.</p>
Tasks	<p>The task(s) for Gradle to execute. A list of tasks can be taken from <code>gradlew tasks</code> issued from a command prompt.</p> <p>See <a href="#">Gradle Build Script Basics</a>.</p>

## JUNIT TEST RESULTS

Publish to VSTS/TFS	Select this option to publish JUnit Test results produced by the Gradle build to VSTS/TFS.
Test Results Files	Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all xml files whose name starts with TEST-.
Test Run Title	Assign a title for the JUnit test case results for this build.

## CODE COVERAGE

Code Coverage Tool	Choose a code coverage tool to determine the code that is covered by the test cases for the build.
--------------------	--

## ADVANCED

Working Directory	Directory on the build agent where the Gradle wrapper will be invoked from. Defaults to the repository root.
-------------------	--

ARGUMENT	DESCRIPTION
Set JAVA_HOME by JDK Version	Choose which JDK level to run Gradle with. Will attempt to find JDK version and assign JAVA_HOME before running Gradle.
Set JAVA_HOME by Path	Directory on build agent where JDK is located.
JDK Architecture	Optionally supply the architecture (x86, x64) of JDK.
<b>CODE ANALYSIS</b>	
Run SonarQube Analysis	Select if you want to run a SonarQube analysis. See <a href="#">The Gradle build task now supports SonarQube analysis</a> .
Run PMD Analysis	Select if you want to perform a <a href="#">PMD static analysis</a> . A build result page for each project is shown on the <b>Artifacts</b> tab of the completed build. See <a href="#">Gradle build task now also supports PMD analysis</a> .
Run Checkstyle Analysis	Select if you want to perform a <a href="#">Checkstyle static analysis</a> . The build summary reports the number of issues found by Checkstyle. Detailed issue logs are available under the build Artifact tab of the build summary. If the Checkstyle analysis is customized, the task only attempts to find the reports and produce a summary.
<b>CONTROL OPTIONS</b>	

## Q&A

### How do I generate a wrapper from my Gradle project?

The Gradle wrapper allows the build agent to download and configure the exact Gradle environment that is checked into the repository without having any software configuration on the build agent itself other than the JVM.

1. Create the Gradle wrapper by issuing the following command from the root project directory where your build.gradle resides:

```
jamal@fabrikam> gradle wrapper
```

2. Upload your Gradle wrapper to your remote repository.

There is a binary artifact that is generated by the gradle wrapper ( located at `gradle/wrapper/gradle-wrapper.jar` ). This binary file is small and doesn't require updating. If you need to change the Gradle configuration run on the build agent, you update the `gradle-wrapper.properties` .

The repository should look something like this:

```
|-- gradle/
  '-- wrapper/
    '-- gradle-wrapper.jar
    '-- gradle-wrapper.properties
|-- src/
|-- .gitignore
|-- build.gradle
|-- gradlew
|-- gradlew.bat
```

## How do I build an Android project?

[Android Build](#)

## Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

## I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

## I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Grunt

9/12/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015.3](#)



The JavaScript Task Runner

## Demands

The build agent must have the following capability:

- Grunt

## Arguments

ARGUMENT	DESCRIPTION
Grunt File Path	Relative path from the repo root to the grunt script that you want to run. The default value is <code>gruntfile.js</code>
Grunt task(s)	(Optional) Space delimited list of tasks to run. If you leave it blank, the default task will run.
ADVANCED	
Arguments	Additional arguments passed to grunt. See <a href="#">Using the CLI</a> . Tip: <code>--gruntfile</code> is not needed. This argument is handled by the Grunt file path argument shown above.
Working directory	Current working directory when the script is run. If you leave it blank, the working directory is the folder where the script is located.
CONTROL OPTIONS	

## Example

See [Sample Gruntfile](#).

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).



# Build: Gulp

9/12/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Node.js streaming task based build system

## Demands

gulp

## Arguments

ARGUMENT	DESCRIPTION
Gulp file path	Relative path from the repo root to the gulp script that you want to run. The default value is <code>gulpfile.js</code>
Gulp task(s)	(Optional) Space delimited list of tasks to run. If you leave it blank, the default task will run.
ADVANCED	
Arguments	Additional arguments passed to gulp. Tip: <code>--gulpfile</code> is not needed. This argument is handled by the Gulp file path argument shown above.
Working directory	Current working directory when the script is run. If you leave it blank, the working directory is the folder where the script is located.
gulp.js location	gulp.js to run. The default value is <code>node_modules/gulp/bin/gulp.js</code>
CONTROL OPTIONS	

## 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.

- Gulp file path: `gulpfile.js`
- Advanced, `gulp.js` location:  
`node_modules/gulp/bin/gulp.js`

## Build a Node.js app

[Build your Node.js app with Gulp](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Index Sources & Publish Symbols

2/15/2018 • 6 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

## NOTE

A symbol server is available with Package Management in **VSTS** 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.



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 step works only:

- For code in Git or TFVC stored in Team Foundation Server (TFS) or VSTS. It does not work for any other type of repository.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Path to symbols folder	The root path that is searched for symbol files using the search patterns supplied in the next input.
Search pattern	<a href="#">File matching pattern(s)</a> (rooted at the path supplied in the previous input) used to discover <code>pdb</code> s that contain symbols.
Index sources	Adds information about the location of the source repository to the symbols. This enables users using these symbols to navigate to the relevant source code.

ARGUMENT	DESCRIPTION
Publish symbols	Publishes symbols to the symbol server selected in the next inputs.
Symbol server type	<p><b>Package Management in Visual Studio Team Services:</b></p> <ul style="list-style-type: none"> <li>Select this option to use the symbol server built into the <a href="#">Package Management extension</a>.</li> </ul> <p><b>File share:</b></p> <ul style="list-style-type: none"> <li>Select this option to use the file share supplied in the next input.</li> </ul>
Path to publish symbols	<p>The path to the SymStore file share.</p> <p>To prepare your SymStore symbol store:</p> <ol style="list-style-type: none"> <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> <p>If you leave this argument blank, your symbols will be source indexed but not published. (You can also store your symbols with your drops. See <a href="#">Publish Build Artifacts</a>).</p>
ADVANCED	
Verbose logging	Enables additional log details.
Warn if not indexed	<p>Enable this option if you want the build summary to show a warning when sources are not indexed for a PDB file. A common cause of sources to not be indexed are when your solution depends on binaries that it doesn't build.</p> <p>Even if you don't select this option, the messages are written in log.</p>
Max wait time (min)	If you want to set a time limit for this step, specify the number of minutes here. The build fails when the limit is reached. If you leave it blank, limit is 2 hours.
Product	If you are publishing your symbols, you can specify the product parameter that is passed to symstore.exe. If blank, <code>\$(Build.DefinitionName)</code> is passed.
Version	If you are publishing your symbols, you can specify the version parameter that is passed to symstore.exe. If blank, <code>\$(Build.BuildNumber)</code> is passed.
Artifact name	Specify the pattern used for the name of the link from the artifact tab in the build summary to the file share where you are publishing your symbols. For example, if you specify <code>Symbols_\$(BuildConfiguration)</code> , then the name of the link to your published release symbols would be <code>Symbols_release</code>
CONTROL OPTIONS	

# Use indexed symbols to debug your app

You can use your indexed symbols to debug an app on a different machine from where the sources were built.

## Enable your dev machine

In Visual Studio you may need to enable the following two options:

- Debug -> Options -> Debugging -> General
  - -> Enable source server support
  - -> Allow source server for partial trust assemblies (Managed only)

## Overriding at debug time

The mapping information injected into the PDB files contains variables that can be overridden at debugging time. Overriding the variables may be required if the collection URL has changed. When overriding the mapping information, the goals are to construct:

- A command (SRCSRVCMD) that the debugger can use to retrieve the source file from the server.
- A location (SRCRVTRG) where the debugger can find the retrieved source file.

The mapping information may look something like the following:

```
SRCSRV: variables -----
TFS_EXTRACT_TARGET=%target%\%var5%\%fnvar%(%var6%)%fnbks1%(%var7%)
TFS_EXTRACT_CMD=tf.exe git view /collection:%fnvar%(%var2%) /teamproject:"%fnvar%(%var3%)"
/repository:"%fnvar%(%var4%)" /commitId:%fnvar%(%var5%) /path:"%var7%" /output:%SRCRVTRG% %fnvar%(%var8%)
TFS_COLLECTION=http://SERVER:8080/tfs/DefaultCollection
TFS_TEAM_PROJECT=93fc2e4d-0f0f-4e40-9825-01326191395d
TFS_REPO=647ed0e6-43d2-4e3d-b8bf-2885476e9c44
TFS_COMMIT=3a9910862e22f442cd56ff280b43dd544d1ee8c9
TFS_SHORT_COMMIT=3a991086
TFS_APPLY_FILTERS=/applyfilters
SRCRVVERCTRL=git
SRCRVERRDESC=access
SRCRVERRVAR=var2
SRCRVTRG=%TFS_EXTRACT_TARGET%
SRCRVCMD=%TFS_EXTRACT_CMD%
SRCSRV: source files -----
C:\BuildAgent\_work\1\src\MyApp\Program.cs*TFS_COLLECTION*TFS_TEAM_PROJECT*TFS_REPO*TFS_COMMIT*TFS_SHORT_COMMIT*/MyApp/Program.cs*TFS_APPLY_FILTERS
C:\BuildAgent\_work\1\src\MyApp\SomeHelper.cs*TFS_COLLECTION*TFS_TEAM_PROJECT*TFS_REPO*TFS_COMMIT*TFS_SHORT_COMMIT*/MyApp/SomeHelper.cs*TFS_APPLY_FILTERS
```

The above example contains two sections: 1) the variables section and 2) the source files section. The information in the variables section is what can be overridden. The variables can leverage other variables, and can leverage information from the source files section.

To override one or more of the variables while debugging with Visual Studio, create an ini file

`%LOCALAPPDATA%\SourceServer\srsrv.ini`. Set the content of the INI file to override the variables. For example:

```
[variables]
TFS_COLLECTION=http://DIFFERENT_SERVER:8080/tfs/DifferentCollection
```

## Q&A

### 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 VSTS (provided you have embedded SourceLink information and enabled it in your IDE).

### **Where can I learn more about symbol stores and debugging?**

[Symbol Server and Symbol Stores](#)

[SymStore](#)

[Use the Microsoft Symbol Server to obtain debug symbol files](#)

[The Srcsrv.ini File](#)

[Source Server](#)

[Source Indexing and Symbol Servers: A Guide to Easier Debugging](#)

### **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

### **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

### **How long are Symbols retained?**

When symbols are published to VSTS they are associated with a build. When the build is deleted either manually or due to retention policy then the symbols are also deleted. If you want to retain the symbols indefinitely then you should mark the build as Retain Indefinately.

# Build: Jenkins Queue Job

9/12/2017 • 3 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017



Queue a job on a Jenkins server

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Jenkins service endpoint	Select the service endpoint for your Jenkins instance. To create one, click <b>Manage</b> and create a new Jenkins Service Endpoint.
Job name	The name of the Jenkins job to queue. This must exactly match the job name on the Jenkins server.
Capture console output and wait for completion	If selected, this step 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 step will successfully complete without waiting for the Jenkins build to run.
Capture pipeline output and wait for pipeline completion	This option is similar to capture console output except it will capture the output for the entire Jenkins pipeline, wait for completion for the entire pipeline, and succeed/fail based on the pipeline result.
Parameterized job	Select this option if the Jenkins job requires parameters.

ARGUMENT	DESCRIPTION
Job parameters	<p>This option is available for parameterized jobs. Specify job parameters, one per line, in the form <b>parameterName=parameterValue</b></p> <p>To set a parameter to an empty value (useful for overriding a default value) leave off the parameter value, e.g. specify <b>parameterName=</b></p> <p>Variables are supported, e.g. to define the <b>commitId</b> parameter to be the <b>git commit ID</b> for the build, use: <b>commitId=\${Build.SourceVersion}</b>.</p> <p>Supported Jenkins parameter types are:</p> <ul style="list-style-type: none"> <li>• Boolean</li> <li>• String</li> <li>• Choice</li> <li>• Password</li> </ul>
Trust server certificate	Selecting this option results in the Jenkins server's SSL certificate being trusted even if it is self-signed or cannot be validated by a Certificate Authority (CA).

## Team Foundation Server Plug-in

You can use Team Foundation Server Plug-in (version 5.2.0 or newer) to automatically collect files from the Jenkins workspace and download them into the build.

To set it up:

1. Install the [Team Foundation Server Plug-in](#) on the Jenkins server.
2. On the Jenkins server, for each job you would like to collect results from, add the **Collect results for TFS/VSTS post-build action** and then configure it with one or more pairs of result type and include file pattern.
3. On the Jenkins Queue Job build task enable the **Capture console output and wait for completion** to collect results from the root level job, or the **Capture pipeline output and wait for pipeline completion** to collect results from all pipeline jobs.

Results will be downloaded to the **\$(Build.StagingDirectory)/jenkinsResults/<Job Name>/team-results.zip** and extracted to this location. Each set of result types collected by the plug-in, will be under the team-results directory, **\$(Build.StagingDirectory)/jenkinsResults/<Job Name>/team-results/<ResultType>/**. This is the directory where build results can be published by downstream tasks (e.g. Publish Test Results, and Publish Code Coverage Results).

## Q&A

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See

[Variables](#).

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Maven

9/13/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



to build your Java code.

## Demands

The build agent must have the following capability:

- Maven

## Arguments

ARGUMENT	DESCRIPTION
Maven POM file	Relative path from the repo root to the Maven POM .xml file. See <a href="#">Introduction to the POM</a> .
Options	Specify any Maven options you want to use.
Goal(s)	In most cases, set this to <code>package</code> to compile your code and package it into a .war file. If you leave this argument blank, the build will fail. See <a href="#">Introduction to the Maven build lifecycle</a> .

### JUNIT TEST RESULTS

Publish to VSTS/TFS	Select this option to publish JUnit Test results produced by the Maven build to VSTS/TFS.
Test Results Files	Test results files path. Wildcards can be used. For example, <code>**/TEST-*.xml</code> for all xml files whose name starts with TEST- ."

### ADVANCED

JDK Version	Will attempt to discover the path to the selected JDK version and set JAVA_HOME accordingly.
JDK Architecture	Optionally supply the architecture (x86, x64) of JDK.

### CODE ANALYSIS

Run SonarQube Analysis	Select if you want to run a SonarQube analysis. See <a href="#">The Maven build task now simplifies SonarQube analysis</a> .
------------------------	--

ARGUMENT	DESCRIPTION
Run PMD Analysis	Select if you want to perform a <a href="#">PMD static analysis</a> . A build result page for each .pom file is shown on the <b>Artifacts</b> tab of the completed build. See <a href="#">The Maven build task now supports PMD analysis out of the box</a> .
CONTROL OPTIONS	

## Example

[Build and Deploy your Java application to an Azure web app](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: MSBuild

11/2/2017 • 4 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**



## Demands

msbuild

**VSTS:** If your team uses Visual Studio 2017 and you want to use our hosted agents, make sure you select as your default queue the **Hosted VS2017**. See [Hosted agents](#).

## Arguments

ARGUMENT	DESCRIPTION
Project	<p>If you want to build a single project, click the ... button and select the project.</p> <p>If you want to build multiple projects, specify search criteria. You can use a single-folder wildcard (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>**\*.*proj</code> searches for all MSBuild project (*.proj) files in all subdirectories.</p> <p>Make sure the projects you specify are downloaded by this build definition. 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 step</a> instead of the MSBuild step.</p>
MSBuild Arguments	You can pass additional arguments to MSBuild. For syntax, see <a href="#">MSBuild Command-Line Reference</a> .
Platform	<p>Specify the platform you want to build such as <code>Win32</code>, <code>x86</code>, <code>x64</code> or <code>any cpu</code>.</p> <p>Tips:</p> <ul style="list-style-type: none"><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
Configuration	<p>Specify the configuration you want to build such as <code>debug</code> or <code>release</code>.</p> <p>Tip: Declare a build variable such as <code>BuildConfiguration</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildConfiguration)</code>. This way you can modify the platform when you queue the build and enable building multiple configurations.</p>
Clean	<p>Set to False if you want to make this an incremental build. This setting might reduce your build time, especially if your codebase is large. This option has no practical effect unless you also set Clean repository to False.</p> <p>Set to True if you want to rebuild all the code in the code projects. This is equivalent to the MSBuild <code>/target:clean</code> argument.</p>
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 step.</p>
<b>ADVANCED</b>	
Record Project Details	Select this checkbox if you want details about how much time was needed to build each project. You can see these details when you select this build step in a completed build.
MSBuild	In some cases you might need more control over the version of MSBuild that you are running.
<b>CONTROL OPTIONS</b>	

## Q&A

### Should I use the Visual Studio Build step or the MSBuild step?

If you are building a solution, in most cases you should use the [Visual Studio Build step](#). This step 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 step](#). For example, you should use it if you are building code projects apart from a solution.

### Where can I learn more about MSBuild?

[MSBuild step](#)

[MSBuild reference](#)

[MSBuild command-line reference](#)

### How do I build multiple configurations for multiple platforms?

1. On the Variables tab, make sure you've got variables defined for your configurations and platforms. To specify multiple values, separate them with commas.

For example, for a .NET app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	any cpu

For example, for a C++ app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	x86, x64

2. On the Options tab select **MultiConfiguration** and specify the Multipliers, separated by commas. For example: `BuildConfiguration, BuildPlatform`

Select Parallel if you want to distribute the jobs (one for each combination of values) to multiple agents in parallel if they are available.

3. On the Build tab, select this step and specify the Platform and Configuration arguments. For example:

- Platform: `$(BuildPlatform)`
- Configuration: `$(BuildConfiguration)`

### Can I build TFSBuild.proj files?

You cannot build TFSBuild.proj files. These kinds of files are generated by TFS 2005 and 2008. These files contain tasks and targets are supported only using [XAML builds](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Visual Studio Build

11/3/2017 • 4 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Build with MSBuild and set the Visual Studio version property.

## Demands

msbuild, visualstudio

**VSTS:** If your team wants to use Visual Studio 2017 with our hosted agents, select **Hosted VS2017** as your default build queue.. See [Hosted agents](#).

## Arguments

ARGUMENT	DESCRIPTION
Solution	<p>If you want to build a single solution, click the ... button and select the solution.</p> <p>If you want to build multiple solutions, specify search criteria. You can use a single-folder wildcard (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>**\*.sln</code> searches for all <code>.sln</code> files in all subdirectories.</p> <p>Make the sure the solutions you specify are downloaded by this build definition. 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 (<code>.*proj</code>) files.</li><li>• If you are building a customized MSBuild project file, we recommend you use the <a href="#">MSBuild step</a> instead of the Visual Studio Build step.</li></ul>
MSBuild Arguments	You can pass additional arguments to MSBuild. For syntax, see <a href="#">MSBuild Command-Line Reference</a> .

ARGUMENT	DESCRIPTION
Platform	<p>Specify the platform you want to build such as <code>Win32</code>, <code>x86</code>, <code>x64</code> or <code>any cpu</code>.</p> <p>Tips:</p> <ul style="list-style-type: none"> <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>
Configuration	<p>Specify the configuration you want to build such as <code>debug</code> or <code>release</code>.</p> <p>Tip: Declare a build variable such as <code>BuildConfiguration</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildConfiguration)</code>. This way you can modify the platform when you queue the build and enable building multiple configurations.</p>
Clean	<p>Set to False if you want to make this an incremental build. This setting might reduce your build time, especially if your codebase is large. This option has no practical effect unless you also set Clean repository to False.</p> <p>Set to True if you want to rebuild all the code in the code projects. This is equivalent to the MSBuild <code>/target:clean argument</code>.</p>
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 step.</p>
Visual Studio Version	<p>To avoid problems overall, you must make sure this value matches the version of Visual Studio used to create your solution.</p> <p>The value you select here adds the  <code>/p:VisualStudioVersion={numeric_visual_studio_version}</code></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>VSTS:</b> If your team wants to use Visual Studio 2017 with our hosted agents, select <b>Hosted VS2017</b> as your default build queue. See <a href="#">Hosted agents</a>.</p>
<b>ADVANCED</b>	

ARGUMENT	DESCRIPTION
MSBuild Architecture	Select either MSBuild x86 or MSBuild x64.  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.
Record Project Details	Select this checkbox if you want details about how much time was needed to build each project. You can see these details when you select this build step in a completed build.
<b>CONTROL OPTIONS</b>	

## Q&A

### Should I use the Visual Studio Build step or the MSBuild step?

If you are building a solution, in most cases you should use the [Visual Studio Build step](#). This step 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 step](#). For example, you should use it if you are building code projects apart from a solution.

### Where can I learn more about MSBuild?

[MSBuild step](#)

[MSBuild reference](#)

[MSBuild command-line reference](#)

### How do I build multiple configurations for multiple platforms?

1. On the Variables tab, make sure you've got variables defined for your configurations and platforms. To specify multiple values, separate them with commas.

For example, for a .NET app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	any cpu

For example, for a C++ app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	x86, x64

2. On the Options tab select **MultiConfiguration** and specify the Multipliers, separated by commas. For example: `BuildConfiguration, BuildPlatform`

Select Parallel if you want to distribute the jobs (one for each combination of values) to multiple agents in parallel if they are available.

3. On the Build tab, select this step and specify the Platform and Configuration arguments. For example:

- Platform: `$(BuildPlatform)`
- Configuration: `$(BuildConfiguration)`

### Can I build TFSBuild.proj files?

You cannot build TFSBuild.proj files. These kinds of files are generated by TFS 2005 and 2008. These files contain tasks and targets are supported only using [XAML builds](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Xamarin.Android

12/4/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Build an Android app with Xamarin

## Demands

AndroidSDK, MSBuild, Xamarin.Android

## Arguments

ARGUMENT	DESCRIPTION
Project	If you want to build a single Xamarin.Android project, click the ... button and select the project.  If you want to build multiple projects, specify search criteria. You can use a single-folder wildcard ( <code>*</code> ) and recursive wildcards ( <code>**</code> ). For example, <code>**/*.Android.csproj</code> searches for all Android.csproj files in all subdirectories in your repo.  Note: The projects must have a PackageForAndroid target.
Target	(Optional) Specify the project targets you want to build. Use a semicolon to separate multiple targets.
Output Directory	Use a <a href="#">variable</a> to specify the folder where you want the output files to go. For example: <code>\$(build.binariesdirectory)/\$(BuildConfiguration)</code>
Configuration	Specify the configuration you want to build such as <code>debug</code> or <code>release</code> .  Tip: Declare a build variable such as <code>BuildConfiguration</code> on the Variables tab (selecting Allow at Queue Time) and reference it here as <code>\$(BuildConfiguration)</code> . This way you can modify the platform when you queue the build and enable building multiple configurations.

### MSBUILD OPTIONS

MSBuild Location	(Optional) Path to MSBuild (on Windows) or xbuild (on macOS). Default behavior is to search for the latest version.
Additional Arguments	You can pass additional arguments to MSBuild (on Windows) or xbuild (on macOS). For syntax, see <a href="#">MSBuild Command-Line Reference</a> .

### JDK OPTIONS

ARGUMENT	DESCRIPTION
Select JDK to use for the build	Pick the JDK to be used during the build by selecting a JDK version that will be discovered during builds or by manually entering a JDK path. <ul style="list-style-type: none"> <li>• <b>JDK Version:</b> Select the JDK version you want to use.</li> <li>• <b>JDK Path:</b> Specify the path to the JDK you want to use.</li> </ul>
JDK Architecture	Select x86 or x64.
<b>CONTROL OPTIONS</b>	

## Example

[Build your Xamarin app](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Xamarin.iOS

12/4/2017 • 1 min to read • [Edit Online](#)

[VSTS | TFS 2018 | TFS 2017 | TFS 2015](#)



Build an iOS app with Xamarin on macOS

## Demands

Xamarin.iOS

## Arguments

ARGUMENT	DESCRIPTION
Solution	Click the ... button and select your solution.
Configuration	Specify a configuration such as <code>Ad-Hoc</code> , <code>AppStore</code> , <code>Debug</code> , or <code>Release</code>
Create App Package	Select if you want to create an .IPA app package file.
Build for iOS Simulator	Select if you want to build for the iOS Simulator instead of for physical iOS devices.
<b>(OPTIONAL) SIGNING &amp; PROVISIONING</b>	
Override Using (Optional)	If the build should use a signing or provisioning method that is different than the default, choose that method here.  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.  Leave the corresponding fields blank if you do not wish to override default build settings.
P12 Certificate File	Relative path to a PKCS12-formatted .p12 certificate file that contains a signing certificate to be used for this build.
P12 Password	Password to the .p12 file.  <b>Important:</b> Use a <a href="#">secret variable</a> to avoid exposing this value.
Provisioning Profile File	Relative path to .mobileprovision file that contains the provisioning profile override to be used for this build.

ARGUMENT	DESCRIPTION
Remove Profile After Build	Select if you want the contents of the provisioning profile file to be removed from the build agent after the build is complete.  <b>Important:</b> Select only if you are running one agent per user.
<b>ADVANCED</b>	
Arguments	(Optional) Specify additional command-line arguments for this build.
Working Directory	Working directory for the build. If you leave it blank, it is the root of the repo.
Xbuild Path	(Optional) Specify the path to <a href="#">xbuild</a> . If you leave it blank, the default xbuild path is used.
<b>CONTROL OPTIONS</b>	

## Example

[Build your Xamarin app](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Xcode

1/17/2018 • 5 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Build, test, or archive an Xcode workspace on macOS. Optionally package an app.

## Demands

xcode

## Arguments

ARGUMENT	DESCRIPTION
Actions	Enter a space-delimited list of actions. Valid options are <code>build</code> , <code>clean</code> , <code>test</code> , <code>analyze</code> , and <code>archive</code> . For example, <code>clean build</code> will run a clean build. See the <a href="#">xcodebuild man page</a> .
Configuration	Enter the Xcode project or workspace configuration to be built. The default value of this field is the variable <code>\$(Configuration)</code> . When using a variable, make sure to specify a value (for example, <code>Release</code> ) on the <a href="#">Variables</a> tab.
SDK	Specify an SDK to use when building the Xcode project or workspace. From the macOS Terminal application, run <code>xcodebuild -showsdk</code> s to display the valid list of SDKs. The default value of this field is the variable <code>\$(SDK)</code> . When using a variable, make sure to specify a value (for example, <code>iphonesimulator</code> ) on the <a href="#">Variables</a> tab.
Workspace or project path	(Optional) Enter a relative path from the root of the repository to the Xcode workspace or project. For example, <code>MyApp/MyApp.xcworkspace</code> or <code>MyApp/MyApp.xcodeproj</code> .
Scheme	(Optional) Enter a scheme name defined in Xcode. It must be a shared scheme, with its <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.
Xcode version	Specify the target version of Xcode. Select <code>Default</code> to use the default version of Xcode on the agent machine. Selecting a version number (e.g. <code>Xcode 9</code> ) relies on environment variables being set on the agent machine for the version's location (e.g. <code>XCODE_9_DEVELOPER_DIR=/Applications/Xcode_9.0.0.app/Contents/Developer</code> ). Select <code>Specify path</code> to provide a specific path to the Xcode developer directory.
Xcode developer path	(Optional) Enter a path to a specific Xcode developer directory (e.g. <code>/Applications/Xcode_9.0.0.app/Contents/Developer</code> ). This is useful when multiple versions of Xcode are installed on the agent machine.
<b>(OPTIONAL) SIGNING &amp; PROVISIONING</b>	

ARGUMENT	DESCRIPTION
Signing style	Choose the method of signing the build. Select <code>Do not code sign</code> to disable signing. Select <code>Project defaults</code> to use only the project's signing configuration. Select <code>Manual signing</code> to force manual signing and optionally specify a signing identity and provisioning profile. Select <code>Automatic signing</code> to force automatic signing and optionally specify a development team ID. If your project requires signing, use the "Install Apple..." tasks to install certificates and provisioning profiles prior to the Xcode build.
Signing identity	(Optional) Enter a signing identity override with which to sign the build. This may require unlocking the default keychain on the agent machine. If no value is entered, the Xcode project's setting will be used.
Provisioning profile UUID	(Optional) Enter the UUID of an installed provisioning profile to be used for this build. Use separate build tasks with different schemes or targets to specify separate provisioning profiles by target in a single workspace (iOS, tvOS, watchOS).
Team ID	(Optional, unless you are a member of multiple development teams.) Specify the 10-character development team ID.
<b>PACKAGE OPTIONS</b>	
Create app package	Indicate whether an IPA app package file should be generated as a part of the build.
Archive path	(Optional) Specify a directory where created archives should be placed.
Export path	(Optional) Specify the destination for the product exported from the archive.
Export options	Select a way of providing options for exporting the archive. When the default value of <code>Automatic</code> is selected, the export method is automatically detected from the archive. Select <code>Plist</code> to specify a plist file containing export options. Select <code>Specify</code> to provide a specific <b>Export method</b> and <b>Team ID</b> .
Export method	Enter the method that Xcode should use to export the archive. For example: <code>app-store</code> , <code>package</code> , <code>ad-hoc</code> , <code>enterprise</code> , or <code>development</code> .
Team ID	(Optional) Enter the 10-character team ID from the Apple Developer Portal to use during export.
Export options plist	Enter the path to the plist file that contains options to use during export.
Export arguments	(Optional) Enter additional command line arguments to be used during export.
<b>DEVICES &amp; SIMULATORS</b>	
Destination platform	Select the destination device's platform to be used for UI testing when the generic build device isn't valid. Choose <code>Custom</code> to specify a platform not included in this list. When <code>Default</code> is selected, no simulators nor devices will be targeted.

ARGUMENT	DESCRIPTION
Destination type	Choose the destination type to be used for UI testing. Devices must be connected to the Mac performing the build via a cable or network connection. See <b>Devices and Simulators</b> in Xcode.
Simulators	Enter an Xcode simulator name to be used for UI testing. For example, enter <code>iPhone X</code> (iOS and watchOS) or <code>Apple TV 4K</code> (tvOS). A target OS version is optional and can be specified in the format ' <code>OS=versionNumber</code> ', such as <code>iPhone X,OS=11.1</code> . A list of simulators installed on the <b>Hosted macOS Preview</b> agent can be <a href="#">found here</a> .
Devices	Enter the name of the device to be used for UI testing, such as <code>Raisa's iPad</code> . Only one device is currently supported. Note that Apple does not allow apostrophes ('') in device names. Instead, right single quotation marks (') can be used.
<b>ADVANCED</b>	
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 the <a href="#">xcodebuild man page</a> .
Working directory	(Optional) Enter the working directory in which to run the build. If no value is entered, the root of the repository will be used.
Output directory	Enter a path relative to the working directory where build output (binaries) will be placed. The default value includes variables. When these are used, make sure to specify values on the <a href="#">Variables</a> tab.
Use xcpretty	Specify whether to use xcpretty to format xcodebuild output and generate JUnit test results. Enabling this requires xcpretty to be installed on the agent machine. It is preinstalled on VSTS hosted build agents. See <a href="#">xcpretty</a> on GitHub.
Publish test results to VSTS/TFS	If xcpretty is enabled above, specify whether to publish JUnit test results to VSTS/TFS.
<b>CONTROL OPTIONS</b>	

## Example

[Build your Xcode app](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Build: Xcode Package iOS

11/14/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



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

## 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 <b>xcdebuild -showsdk</b> 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> .
Path to place .ipa	Relative path where the .ipa will be placed. The directory will be created if it doesn't exist. The default value is \$(SDK)/\$(Configuration)/build.sym/\$(Configuration)- \$(SDK)/output . Make sure to specify the variable values on the <a href="#">variables tab</a> .

## CONTROL OPTIONS

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Archive Files

9/12/2017 • 2 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017



Create an archive file from a source folder. A variety of standard archive formats are supported including: .zip, .jar, .war, .ear, .tar, .7z, and others.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Root folder (or file) to archive	<p>The folder (or file) you wish to archive. The default file path is relative from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ).</p> <p>If the specified path is a folder, recursively, all nested files and folders will be included in the archive.</p> <p>If the specified path is a file, only the single file will be included in the archive.</p>
Prefix root folder name to archive paths	<p>If selected, the root folder name will be prefixed to file paths within the archive. Otherwise, all file paths will start one level lower.</p> <p>For example, suppose the selected root folder is: <code>/home/user/output/classes/</code>, and contains: <code>com/acme/Main.class</code>.</p> <ul style="list-style-type: none"><li>• If selected, the resulting archive would contain: <code>classes/com/acme/Main.class</code>.</li><li>• Otherwise, the resulting archive would contain: <code>com/acme/Main.class ..</code></li></ul>
Archive type	<p>Specify the compression scheme used. To create <code>foo.jar</code>, for example, choose <code>zip</code> for the compression, and specify <code>foo.jar</code> as the archive file to create. For all tar files (including compressed ones), choose <code>tar</code>.</p> <ul style="list-style-type: none"><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>

ARGUMENT	DESCRIPTION
Tar compression	<p>Only applicable if the <code>tar</code> archive type is selected.</p> <p>Optionally choose a compression scheme, or choose <code>None</code> to create an uncompressed tar file.</p> <ul style="list-style-type: none"> <li>• <code>gz</code> - default, gzip compression (<code>.tar.gz</code>, <code>.tar.tgz</code>, <code>.taz</code>)</li> <li>• <code>bz2</code> - bzip2 compression (<code>.tar.bz2</code>, <code>.tz2</code>, <code>.tbz2</code>)</li> <li>• <code>xz</code> - xz compression (<code>.tar.xz</code>, <code>.txz</code>)</li> <li>• <code>None</code> - no compression, choose this to create a uncompressed tar file (<code>.tar</code>)</li> </ul>
Archive file to create	<p>Specify the name of the archive file to create. The file extension should match the selected archive type. For example to create <code>foo.tgz</code>, select the <code>tar</code> archive type, <code>gz</code> for tar compression.</p>
Replace existing archive	<p>If an existing archive exists, specify whether to overwrite it. Otherwise, files will be added to it as long as it is not a compressed tar.</p> <p>If adding to an existing archive, these types are supported:</p> <ul style="list-style-type: none"> <li>• <code>zip</code></li> <li>• <code>7z</code></li> <li>• <code>tar</code> - uncompressed only</li> <li>• <code>wim</code></li> </ul>

#### CONTROL OPTIONS

## Q & A

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Azure function

1/31/2018 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015.3](#)



Invoke a HTTP triggered function in an Azure function app and parse the response.

## Demands

Can be used in only an [agentless phase](#) of a release definition.

## Arguments

PARAMETER	COMMENTS
<b>Azure function URL</b>	Required. The URL of the Azure function to be invoked.
<b>Function key</b>	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.
<b>Headers</b>	Optional. The header in JSON format to be attached to the request sent to the function.
<b>Request body</b>	Optional. The request body for the Azure function call.
<b>Execution mode</b>	Required. <b>Synchronous mode</b> (the default), or <b>Asynchronous call</b> where the Azure function calls back to update the timeline record.
<b>Response parse expression</b>	Optional. How to parse the response body for success.
<b>Control options</b>	See <a href="#">Control options</a>

Succeeds if the function returns success and the response body parsing is successful.

For more information about using this task, see [Approvals and gates overview](#).

Also see this task on [GitHub](#).

## Q & A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Azure monitor

1/31/2018 • 1 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015.3**



Observe the configured Azure monitor rules for active alerts.

Can be used in only an [agentless phase](#) of a release definition.

None

## Arguments

PARAMETER	COMMENTS
<b>Azure subscription</b>	Required. Select an Azure Resource Manager service endpoint.
<b>Resource group name</b>	Required. The resource group in the subscription containing the monitor functions.
<b>Resource type(s)</b>	Optional. Select the resource type in the selected group. Leave empty to use all the resource types in the resource group.
<b>Resource(s)</b>	Optional. Select the resources of the chosen types in the selected group. Leave empty to use all matching resources.
<b>Select alert rules</b>	Required. Select from the currently configured alert rules in the resource group. Filter the displayed list for selected resource types and resources.
<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](#).

## Q & A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Batch script

9/12/2017 • 2 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**



Run a Windows .bat or .cmd script and optionally allow it to change the environment.

## Arguments

ARGUMENT	DESCRIPTION
Path	Specify the path to the .bat or .cmd script you want to run. The path must be a fully qualified path or a valid path relative to the default working directory. In Team Foundation Build, this directory is <a href="#">\$(Build.SourcesDirectory)</a> .
Arguments	Specify arguments to pass to the script.
Modify environment	Select this check box if you want environment variable modifications in the script to affect subsequent tasks.
ADVANCED	
Working folder	Specify the working directory in which you want to run the script. If you leave it empty, the working directory is the folder where the script is located.
Fail on standard error	Select this check box if you want the build to fail if errors are written to the StandardError stream.
CONTROL OPTIONS	

## Example

Create `test.bat` at the root of your repo:

```
@echo off
echo Hello World from %AGENT_NAME%.
echo My ID is %AGENT_ID%.
echo AGENT_WORKFOLDER contents:
@dir %AGENT_WORKFOLDER%
echo AGENT_BUILDDIRECTORY contents:
@dir %AGENT_BUILDDIRECTORY%
echo BUILD_SOURCESDIRECTORY contents:
@dir %BUILD_SOURCESDIRECTORY%
echo Over and out.
```

On the Build tab of a build definition, add this step:



## Utility: Batch Script

Run test.bat.

- Path: `test.bat`

# Q&A

## Where can I learn about batch files?

[Using batch files](#)

## Where can I learn Windows commands?

[An A-Z Index of the Windows CMD command line](#)

## How do I set a variable so that it can be read by subsequent scripts and tasks?

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

## Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

## Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

## Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

## I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

## I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Command line

9/12/2017 • 1 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**

 Run a program from the command prompt.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Tool	Specify the tool you want to run.  If you are using an on-premises agent, in most cases you should configure the machine so that the tool is on the PATH environment variable. But if you know the location of the tool, you can specify a fully qualified path.
Arguments	Specify arguments to pass to the tool.
ADVANCED	
Working folder	Specify the working directory in which you want to run the command. If you leave it empty, the working directory is <code>\$(Build.SourcesDirectory)</code> .
Fail on standard error	Select this check box if you want the build to fail if errors are written to the StandardError stream.
CONTROL OPTIONS	

## Example

On the Build tab of a build definition, add these steps:

 <b>Utility: Command Line</b>	Get the date. <ul style="list-style-type: none"><li>Tool: <code>date</code></li><li>Arguments: <code>/t</code></li></ul>
 <b>Utility: Command Line</b>	Display the operating system version. <ul style="list-style-type: none"><li>Tool: <code>ver</code></li></ul>



#### Utility: Command Line

Display the environment variables.

- Tool: `set`



#### Utility: Command Line

Display all files in all the folders created by the build definition.

- Tool: `dir`
- Arguments: `/s`
- Advanced, Working folder:  
`$(Agent.BuildDirectory)`

## Q&A

### Where can I learn Windows commands?

[An A-Z Index of the Windows CMD command line](#)

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Copy and Publish Build Artifacts

12/19/2017 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2015.3 and newer](#) | [TFS 2015 RTM](#) | [Previous versions \(XAML builds\)](#)



Copy build artifacts to a staging folder and then publish them to the server or a file share.

## IMPORTANT

Are you using Visual Studio Team Services (VSTS), Team Foundation Server (TFS) 2015.3 or newer? If so, then we recommend that you do not use this task; it's deprecated. Instead, you should use the **Copy Files** and **Publish Build Artifacts** tasks. See [Artifacts in Team Build](#).

You should use this task only if you're using Team Foundation Server (TFS) 2015 RTM. In that version of TFS this task is listed under the **Build** category and it's called **Publish Build Artifacts**.

Files are copied to the `$(Build.ArtifactStagingDirectory)` staging folder and then published.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Copy Root	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 build agent working directory.
Contents	Specify pattern filters (one on each line) that you want to apply to the list of files to be copied. For example: <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>
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 Team Build</a> .
CONTROL OPTIONS	

## Q & A

**Q: This step didn't produce the outcome I was expecting. How can I fix it?**

This step 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 step](#) and the [Publish Build Artifacts step](#).

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Copy Files

9/12/2017 • 2 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015.3**



Copy files from a source folder to a target folder using match patterns.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Source Folder	<p>Folder that contains the files you want to copy. If you leave it empty, the copying is done from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ).</p> <p>If your build produces artifacts outside of the sources directory, specify <code>\$(Agent.BuildDirectory)</code> to copy files from the directory created for the definition.</p>
Contents	<p>Specify match pattern filters (one on each line) that you want to apply to the list of files to be copied. For example:</p> <ul style="list-style-type: none"><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 all files recursively from any <code>bin</code> 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>.</p> <p>More examples are shown below.</p>
Target Folder	Folder where the files will be copied. In most cases you specify this folder using a variable. For example, specify <code>\$(Build.ArtifactStagingDirectory)</code> if you intend to <a href="#">publish the files as build artifacts</a> .
ADVANCED	
Clean Target Folder	Select this check box to delete all existing files in the target folder before beginning to copy.
Over Write	Select this check box to replace existing files in the target folder.

ARGUMENT	DESCRIPTION
CONTROL OPTIONS	

## 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
```

On the Variables tab, `$(BuildConfiguration)` is set to `release`.

#### Arguments

- Source Folder: `$(Build.SourcesDirectory)`
- Contents (example of multiple match patterns):

```
ConsoleApplication1\ConsoleApplication1\bin\**\*.exe
ConsoleApplication1\ConsoleApplication1\bin\**\*.dll
ConsoleApplication1\readme.txt
```

- Contents (example of OR condition):

```
ConsoleApplication1\ConsoleApplication1\bin\**\?(*.exe|*.dll)
ConsoleApplication1\readme.txt
```

- Contents (example of NOT condition):

```
ConsoleApplication1\**\bin\**\!(*.pdb|*.config)
!ConsoleApplication1\**\ClassLibrary\**
ConsoleApplication1\readme.txt
```

- Target Folder: `$(Build.ArtifactStagingDirectory)`

#### Results

These files are copied to the staging directory:

```
-- ConsoleApplication1
|-- readme.txt
`-- ConsoleApplication1
    `-- bin
        `-- Release
            |-- ClassLibrary1.dll
            |-- ClassLibrary2.dll
            |-- ConsoleApplication1.exe
```

# Q & A

## Where can I learn more about file matching patterns?

File matching patterns reference

## How do I use this task to publish artifacts?

See [Artifacts in Team Build](#).

## Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

## Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

## Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

## I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

## I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: cURL Upload Files

9/12/2017 • 1 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017 | TFS 2015



Use [cURL](#) to upload files with supported protocols. (FTP, FTPS, SFTP, HTTP, and more)

## Demands

curl

## Arguments

ARGUMENT	DESCRIPTION
Files	If you want to upload a single file, click the ... button and select the file.  If you want to upload multiple files, specify a minimatch pattern filter. For example, specify <code>**\*.zip</code> to upload all ZIP files in all sub-folders.
Username	Specify the username for server authentication.
Password	Specify the password for server authentication.  <b>Important:</b> Use a <a href="#">secret variable</a> to avoid exposing this value.
URL	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 <code>DICT://</code> , <code>FILE://</code> , <code>FTP://</code> , <code>FTPS://</code> , <code>GOPHER://</code> , <code>HTTP://</code> , <code>HTTPS://</code> , <code>IMAP://</code> , <code>IMAPS://</code> , <code>LDAP://</code> , <code>LDAPS://</code> , <code>POP3://</code> , <code>POP3S://</code> , <code>RTMP://</code> , <code>RTSP://</code> , <code>SCP://</code> , <code>SFTP://</code> , <code>SMTP://</code> , <code>SMTPS://</code> , <code>TELNET://</code> , and <code>TFTP://</code> .
Optional Arguments	Arguments to pass to cURL.
ADVANCED	
Redirect Standard Error to Standard Out	In most cases you should leave this selected.  Select if you want to add <code>--stderr -</code> as an argument to cURL. Otherwise, if you clear this check box, cURL will write its progress bar to stderr, which is interpreted by the build process as error output, which could cause the build to fail.

ARGUMENT	DESCRIPTION
<b>CONTROL OPTIONS</b>	

## Q&A

### Where can I learn more about file matching patterns?

[File matching patterns reference](#)

### Where can I learn FTP commands?

[List of raw FTP commands](#)

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Delay

1/19/2018 • 1 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015.3**

 Pause execution of the process for a fixed delay time.

## Demands

Can be used in only an [agentless phase](#) of a release definition.

## Arguments

PARAMETER	COMMENTS
<b>Display name</b>	Required. The name to display for this task.
<b>Delay Time (minutes)</b>	Required. The number of minutes to delay execution.
<b>Control options</b>	See <a href="#">Control options</a>

Also see this task on [GitHub](#).

## Q & A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Delete files

9/12/2017 • 1 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017 | TFS 2015.3



Delete files or folders.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Source Folder	<p>Folder that contains the files you want to delete. If you leave it empty, the deletions are done from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ).</p> <p>If your build produces artifacts outside of the sources directory, specify <code>\$(Agent.BuildDirectory)</code> to delete files from the build agent working directory.</p>
Contents	<p>Specify minimatch pattern filters (one on each line) that you want to apply to the list of files to be deleted. For example:</p> <ul style="list-style-type: none"><li>• <code>**</code> deletes all files and folders in the root folder.</li><li>• <code>temp</code> deletes the temp 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 temp.</li><li>• <code>**\temp\**</code> deletes all files in any sub-folder named temp.</li><li>• <code>**\temp*</code> deletes any file or folder with a name that begins with temp.</li><li>• <code>**\temp*\**</code> deletes files in any sub-folder that begins with the name temp.</li></ul>
CONTROL OPTIONS	

## Q & A

**Q: What's a minimatch pattern? How does it work?**

A: See:

- <https://github.com/isaacs/minimatch>
- <https://realguess.net/tags/minimatch/>

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.

2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Download Secure File

1/19/2018 • 1 min to read • [Edit Online](#)

## VSTS



Download a secure file to a temporary location on the build or release agent

Use this task to download a [secure file](#) from the server during a build or release.

## Arguments

ARGUMENT	DESCRIPTION
Secure File	Select the secure file to download to a temporary location on the agent. The file will be cleaned up after the build or release.

# Utility: Extract Files

9/12/2017 • 2 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017



Extract files from archives to a target folder using match patterns. A variety of standard archive formats are supported including: .zip, .jar, .war, .ear, .tar, .7z, and others.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Archive file patterns	<p>The archives you want to extract. The default file path is relative from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ).</p> <p>Specify match pattern filters (one on each line) that you want to apply to identify the list of archives to extract. For example:</p> <ul style="list-style-type: none"><li>• <code>test.zip</code> extracts the test.zip file to 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> ).
Clean destination folder before extracting	Select this check box to delete all existing files in the destination folder before beginning to extract archives.
CONTROL OPTIONS	

## Q & A

### Where can I learn more about file matching patterns?

[File matching patterns reference](#)

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.

2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: FTP Upload

9/12/2017 • 2 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017



Upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
FTP Service Endpoint	Select the service endpoint for your FTP server. To create one, click the Manage link and create a new Generic Service Endpoint, enter the FTP server URL for the server URL, e.g. <code>ftp://server.example.com</code> , and required credentials.  Secure connections will always be made regardless of the specified protocol ( <code>ftp://</code> or <code>ftps://</code> ) if the target server supports FTPS. To allow only secure connections, use the <code>ftps://</code> protocol, e.g. <code>ftps://server.example.com</code> . Connections to servers not supporting FTPS will fail if <code>ftps://</code> is specified.
Source folder	The source folder to upload files from. The default file path is relative from the root folder of the repo (same as if you had specified <code>\$(Build.SourcesDirectory)</code> ).
File patterns	File paths or patterns of the files to upload. Supports multiple lines of match patterns. To upload the entire folder content recursively, specify <code>**</code> .
Remote directory	Upload files to this directory on the remote FTP server.
Clean remote directory	Recursively delete all contents of the remote directory before uploading.
Overwrite	Overwrite existing files in the remote directory.
Trust server certificate	Selecting this option results in the FTP server's SSL certificate being trusted with <code>ftps://</code> , even if it is self-signed or cannot be validated by a Certificate Authority (CA).
CONTROL OPTIONS	

## Q & A

**Where can I learn more about file matching patterns?**

## File matching patterns reference

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Install Apple Certificate

11/15/2017 • 1 min to read • [Edit Online](#)

VSTS | TFS 2018



Install an Apple certificate 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

## Arguments

ARGUMENT	DESCRIPTION
Certificate (P12)	Select the certificate (.p12) that was uploaded to <b>Secure Files</b> 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 <b>Variables</b> 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 <b>Variables</b> 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.
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.

# Utility: Install Apple Provisioning Profile

11/15/2017 • 1 min to read • [Edit Online](#)

VSTS | TFS 2018



Install an Apple provisioning profile 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.
- (**VSTS**) Committed to the source repository or copied to a local path on the macOS agent. We recommend encrypting the provisioning profiles if you are committing them to the source repository. The **Decrypt File** task can be used to decrypt them during a build or release.

## Demands

xcode

## Arguments

ARGUMENT	DESCRIPTION
Provisioning Profile Location ( <b>VSTS</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.

# Utility: Invoke HTTP REST API

1/19/2018 • 1 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015.3**



Invoke an HTTP API and parse the response.

## Demands

Can be used in only an [agentless phase](#) of a release definition.

## Arguments

PARAMETER	COMMENTS
<b>Generic endpoint</b>	Required. Select a Generic service endpoint.
<b>Method</b>	Required. 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 function.
<b>Body</b>	Optional. The request body for the function call.
<b>Execution mode</b>	Required. <b>Synchronous mode</b> (the default), or <b>Asynchronous call</b> where the function calls back to update the timeline record.
<b>Response parse expression</b>	Optional. How to parse the response body for success.
<b>Control options</b>	See <a href="#">Control options</a>

Succeeds if the function returns success and the response body parsing is successful.

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.

Fabrikam

Save + Release View releases ...

Pipeline Tasks Variables Retention Options History

Dev Deployment process

Run on agent Run on agent

Deploy Azure App Service Azure App Service Deploy

Quick Web Performance... Cloud-based Web Performance Test

Agentless phase Run on server

Invoke REST API: POST Invoke REST API

Invoke REST API ⓘ

Version 0.\*

Display name \* Invoke REST API: POST

Generic Endpoint \* ⓘ MyGenericConnection

Method \* ⓘ POST

Headers ⓘ { "Content-Type": "application/json" }

Body { "count": 19, "value": [ ] }

Wait For Completion ⓘ

Control Options

The screenshot shows the Azure DevOps Pipeline editor interface. On the left, there's a tree view of pipeline phases: 'Dev' (Deployment process), 'Run on agent' (Run on agent), 'Deploy Azure App Service' (Azure App Service Deploy), 'Quick Web Performance...' (Cloud-based Web Performance Test), 'Agentless phase' (Run on server), and 'Invoke REST API: POST' (Invoke REST API). The 'Invoke REST API: POST' task is highlighted with a red box. To the right of the tree view, the task configuration pane is open, showing details like 'Display name' (Invoke REST API: POST), 'Generic Endpoint' (MyGenericConnection), 'Method' (POST), 'Headers' (Content-Type: application/json), and 'Body' (a JSON object with count: 19 and value: []). There are also 'Control Options' and a 'Wait For Completion' checkbox.

For more information about using this task, see [Approvals and gates overview](#).

Also see this task on [GitHub](#).

## Q & A

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Manual intervention

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

**VSTS | TFS 2018 | TFS 2017 | TFS 2015.3**

 **Pause** Pause an active deployment within an environment, typically to perform some manual steps or actions, and then continue the automated deployment steps.

## Demands

Can be used in only an [agentless phase](#) of a release definition.

## Arguments

PARAMETER	COMMENTS
<b>Display name</b>	Required. The name to display for this task.
<b>Instructions</b>	Optional. The instruction text to display to the user when the task is activated.
<b>Notify users</b>	Optional. The list of users that will be notified that the task has been activated.
<b>On timeout</b>	Required. The action to take (reject or resume) if the task times out with no manual intervention. The default is to reject the deployment.
<b>Control options</b>	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 an environment, typically to perform some manual steps or actions, and then continue the automated deployment steps. 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 phase. 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.

Fabrikam

Save Release View releases ...

Pipeline Tasks Variables Retention Options History

Dev Deployment process

Manual Intervention ⓘ

Version 8.\*

Run on agent

Deploy Azure App Service

Quick Web Performance...

Agentless phase

Manual Intervention

Display name \*

Manual Intervention

Instructions ⓘ

Ready to deploy to \$(Release.EnvironmentName) for customer \$(customerName). Please contact customer to confirm deployment requirements have been met.

Notify users ⓘ

Mateo Escobedo Search users and groups

On timeout ⓘ

Reject  Resume

Control Options

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 environment can resume or reject the manual intervention.

For more information about using this task, see [Approvals and gates overview](#).

Also see this task on [GitHub](#).

## Q & A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: PowerShell

9/12/2017 • 3 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**



Run a PowerShell script

## Demands

DotNetFramework

## Arguments

ARGUMENT	DESCRIPTION
Script filename	Specify the path to the script to you want to run. The path must be a fully qualified path or a valid path relative to the default working directory. In Team Foundation Build, this directory is <a href="#">\$(Build.SourcesDirectory)</a> .
Arguments	Specify arguments to pass to the script. You can use ordinal or named parameters.
Advanced - Working folder	Specify the working directory in which you want to run the script. If you leave it empty, the working directory is the folder where the script is located.
Control options	

## Examples

### Hello World

Create `test.ps1` at the root of your repo:

```
Write-Host "Hello World from $Env:AGENT_NAME."
Write-Host "My ID is $Env:AGENT_ID."
Write-Host "AGENT_WORKFOLDER contents:"
gci $Env:AGENT_WORKFOLDER
Write-Host "AGENT_BUILDDIRECTORY contents:"
gci $Env:AGENT_BUILDDIRECTORY
Write-Host "BUILD_SOURCESDIRECTORY contents:"
gci $Env:BUILD_SOURCESDIRECTORY
Write-Host "Over and out."
```

On the Build tab of a build definition, add this step:

TASK	ARGUMENTS
	Run test.ps1. <b>Script filename:</b> <code>test.ps1</code>

## Write a warning



Set warning message

- Arguments

```
"You've been warned by"
```

- Script

```
Write-Host "$('##vso[task.setvariable variable=WarningMessage]"') $($args[0])"
```



Write warning using task.LogIssue

- Script

```
# Writes a warning to build summary and to log in yellow text  
Write-Host "$('##vso[task.logissue type=warning;]") $($env:WarningMessage) $($task.LogIssue Team Build logging command.)"
```



Write warning using PowerShell command

- Script

```
# Writes a warning to log preceded by "WARNING: "  
Write-Warning $($env:WarningMessage) $($task.LogIssue Team Build logging command.)"
```

## Write an error



Set error message

- Arguments

```
"something went wrong."
```

- Script

```
Write-Host "$('##vso[task.setvariable variable=ErrorMessage]"') $($args[0])"
```



Write error using task.LogIssue

- Script

```
# Writes an error to the build summary and to the log in red text  
Write-Host "$('##vso[task.logissue type=error;]") $($task.LogIssue Team Build logging command reported that") $($env:ErrorMessage)"
```

**TIP**

If you want this error to fail the build, then add this line:

```
exit 1
```



Write error using PowerShell command

- Script

```
# Writes an error to the build summary and the log with details about the error  
Write-Error "$(the Write-Error PowerShell command reported that) $($env:ErrorMessage)"
```

**TIP**

If you don't want this error to fail the build, then clear the **Advanced: Fail on Standard Error** check box.

## ApplyVersionToAssemblies.ps1

[Use a script to customize your build process](#)

## Q&A

### Where can I learn about PowerShell scripts?

[Scripting with Windows PowerShell](#)

[Microsoft Script Center \(the Scripting Guys\)](#)

[Windows PowerShell Tutorial](#)

[PowerShell.org](#)

### How do I set a variable so that it can be read by subsequent scripts and tasks?

[Define and modify your build variables in a script](#)

[Define and modify your release variables in a script](#)

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Publish Build Artifacts

10/12/2017 • 1 min to read • [Edit Online](#)

[VSTS | TFS 2018 | TFS 2017 | TFS 2015.3 | TFS 2015 RTM | Previous versions \(XAML builds\)](#)



Publish build artifacts to Visual Studio Team Services/TFS or a file share.

## TIP

Looking to get started working with build artifacts? See [Artifacts in Team Build](#).

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Path to publish	Path to the folder or file you want to publish. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. Typically, you'll specify <code>\$(Build.ArtifactStagingDirectory)</code> . See <a href="#">Artifacts in Team Build</a> .
Artifact name	Specify the name of the artifact that you want to create. It can be whatever you want. For example: <code>drop</code>
Artifact publish location	In most cases, VSTS/TFS ( <code>Server</code> on <b>TFS 2018 RTM</b> and older) is the best and simplest option. Otherwise, choose a file share and then specify a few more arguments (see below). To learn more, see <a href="#">Artifacts in Team Build</a> .
File share path	Specify the path to the file share where you want to copy the files. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. Publishing artifacts from a Linux or macOS agent to a file share is not supported.
Parallel copy ( <b>VSTS, TFS 2018</b> , or newer)	Select whether to copy files in parallel using multiple threads for greater potential throughput. If this setting is not enabled, a single thread will be used.
Parallel count ( <b>VSTS, TFS 2018</b> , or newer)	Enter the degree of parallelism (the number of threads) used to perform the copy. The value must be at least 1 and not greater than 128. Choose a value based on CPU capabilities of the build agent. Typically, 8 is a good starting value.
Control options	

## Q & A

**Q: I'm having problems. How can I troubleshoot them?**

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

**Q: How do variables work? What variables are available for me to use in the arguments?**

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

# Utility: Publish To Azure Service Bus

1/19/2018 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015.3](#)



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 phase](#) of a release definition.

## Arguments

PARAMETER	COMMENTS
<b>Display name</b>	Required. The name to display for this task.
<b>Azure Service Bus Connection</b>	Required. An existing service connection to an Azure Service Bus.
<b>Message body</b>	Required. The text of the message body to send to the Service Bus.
<b>Wait for Task Completion</b>	Optional. Set this option to force the task to halt until a response is received.
<b>Control options</b>	See <a href="#">Control options</a>

Also see this task on [GitHub](#).

## Q & A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Query Work Items

1/31/2018 • 1 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015.3**

 Ensure the number of matching items returned by a work item query is within the configured thresholds.

Can be used in only an [agentless phase](#) of a release definition.

None

## Arguments

PARAMETER	COMMENTS
<b>Project Name</b>	Required. Select the project in the current VSTS or TFS account. Defaults to the current project.
<b>Query</b>	Required. Select a work item query within the project. Can be a built-in or custom query.
<b>Threshold (max) number of results</b>	Required. Default value = 0
<b>Threshold (min) number of results</b>	Required. Default value = 0
<b>Tags to filter results on</b>	Optional. A list of tags that must be present on the work items.
<b>Control options</b>	See <a href="#">Control options</a>

Succeeds if  $\text{minimum-threshold} \leq \#-\text{matching}-\text{workitems} \leq \text{maximum-threshold}$

For more information about using this task, see [Approvals and gates overview](#).

Also see this task on [GitHub](#).

## Q & A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Service Fabric PowerShell

1/19/2018 • 1 min to read • [Edit Online](#)



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.

## Arguments

ARGUMENT	DESCRIPTION
<b>Cluster Connection</b>	The Azure Service Fabric service endpoint 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/**/dock/compose.yml</code> . <b>Note:</b> 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](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.

- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Utility: Shell script

9/12/2017 • 2 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**



Run a shell script using bash

## Demands

sh

## 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 step to fail if any errors are written to the StandardError stream.
CONTROL OPTIONS	

## Example

Create `test.sh` at the root of your repo:

```
#!/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 "BUILD_SOURCESDIRECTORY is $BUILD_SOURCESDIRECTORY"
echo "BUILD_SOURCESDIRECTORY contents:"
ls -1 $BUILD_SOURCESDIRECTORY
echo "Over and out."
```

On the [Build tab](#) of a build definition, add this step:



## Utility: Shell Script

Run test.bat.

- Script Path: `test.sh`

# Q&A

## Where can I learn about Bash scripts?

[Beginners/BashScripting](#) to get started.

[Awesome Bash](#) to go deeper.

## How do I set a variable so that it can be read by subsequent scripts and tasks?

Define and modify your build variables in a script

Define and modify your release variables in a script

### Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

### Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See [Variables](#).

## Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

## I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

## I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Update Service Fabric App Versions

9/12/2017 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#)



Automatically updates the versions of a packaged Service Fabric application.

This task appends a version suffix to all service and application versions, specified in the manifest files, in an Azure Service Fabric application package.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Application Package	<p>The location of the Service Fabric application package to be deployed to the cluster.</p> <ul style="list-style-type: none"><li>Example: \$(system.defaultworkingdirectory)/**/drop/applicationpackage</li><li>Can include wildcards and variables.</li></ul>
Version Value	<p>The value appended to the versions in the manifest files. Default is .\$(Build.BuildNumber) .</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 \$(VersionSuffix) defined in a PowerShell task:</p> <pre>\$versionSuffix = ".\${[DateTimeOffset]::UtcNow.ToString('yyyyMMdd.HHmmss')}"  Write-Host "##vso[task.setvariable variable=VersionSuffix;]\$versionSuffix"</pre>
Version Behavior	<p>Specify whether to append the version value to existing values in the manifest files, or replace them.</p>
Update only if changed	<p>Select this check box if you want to append the new version suffix to only the packages that have changed from a previous build. If no changes are found, the version suffix from the previous build will be appended.</p> <p><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>

ARGUMENT	DESCRIPTION
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](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Utility: Xamarin license

9/12/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Activate or deactivate Xamarin licenses

## Deprecated

We're deprecating this task because you no longer need a Xamarin license to [build your Xamarin app](#). We recommend that you remove this task from your build to avoid disruption when we remove the task from the product.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Action	Select:  <b>Activate</b> for the first instance of this build step, which should come before any instances of the Xamarin.Android step or the Xamarin.iOS steps.  <b>Deactivate</b> for the second instance of this build step, which should come after all instances of the Xamarin.Android and Xamarin.iOS steps. You should also select <b>Always run</b> under <b>Control options</b> for the last instance of the Xamarin license step.
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 step that you're running in this build definition, such as <b>Xamarin.Android</b> or <b>Xamarin.iOS</b> .
Advanced - Timeout in Seconds	Specify how long you want to allow the build step to wait for the activation or deactivation.
Control options	

## Example

[Build your Xamarin app](#)

# Q&A

## **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

## **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

## **I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

4 min to read •

# Test: Cloud-based Load Test

1/19/2018 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Runs a load test in the cloud with VSTS.

Use this task to understand, test, and validate your app's performance. The task can be used in a build or release definition to trigger a load test by using the VSTS Cloud-based Load Test Service. The Cloud-based Load Test Service is based in Microsoft Azure and can be used to test your app's performance by generating load on it.

## Demands

The build agent must have the following capabilities:

- MSBuild
- Azure PowerShell

## Arguments

ARGUMENT	DESCRIPTION
<b>VSTS connection</b>	The name of a Generic Service Endpoint that references the VSTS account you will be running the load test from and publishing the results to. - Required for builds and releases on TFS and must specify a connection to the VSTS account where the load test will run. - Optional for builds and releases on VSTS. In this case, if not provided, the current VSTS connection is used. - See <a href="#">Generic service endpoint</a> .
<b>Test settings file</b>	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.
<b>Load test files folder</b>	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.
<b>Load test file</b>	Required. The name of the load test file (such as <b>myfile.loadtest</b> ) 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.
<b>Number of permissible threshold violations</b>	Optional. The number of critical violations that must occur for the load test to be deemed unsuccessful, aborted, and marked as failed.
<b>Control options</b>	See <a href="#">Control options</a>

## Examples

- [Load test your app in the cloud](#)
- [Scheduling Load Test Execution](#)

## More Information

- [Cloud-based Load Testing](#)
- [Source code for this task](#)
- [Build your Visual Studio solution](#)
- [Cloud-based Load Testing Knowledge Base](#)

## Related tasks

- [Cloud-based Web Performance Test](#)

## Q&A

### How do I use a Test Settings file?

The **Test settings file** references any setup and cleanup scripts required to execute the load test. For more details see: [Using Setup and Cleanup Script in Cloud Load Test](#)

### When should I specify the number of permissible threshold violations?

Use the **Number of permissible threshold violations** setting if your load test is not already configured with information about how many violations will cause a failure to be reported. For more details, see: [How to: Analyze Threshold Violations Using the Counters Panel in Load Test Analyzer](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- Report problems through the [Developer Community](#)
- Send suggestions on [UserVoice](#)

# Test: Cloud-based Web Performance Test

1/19/2018 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Runs the Quick Web Performance Test with VSTS.

The task can be used in a build or release definition to generate load against an application URL using the VSTS Cloud-based Load Test Service. The Cloud-based Load Test Service is based in Microsoft Azure and can be used to test your app's performance by generating load on it, and verify it exists and is responsive.

## Demands

The build agent must have the following capabilities:

- MSBuild
- Azure PowerShell

## Arguments

ARGUMENT	DESCRIPTION
<b>VSTS connection</b>	The name of a Generic Service Endpoint that references the VSTS account you will be running the load test from and publishing the results to. - Required for builds and releases on TFS and must specify a connection to the VSTS account where the load test will run. - Optional for builds and releases on VSTS. In this case, if not provided, the current VSTS connection is used. - See <a href="#">Generic service endpoint</a> .
<b>Website Url</b>	Required. The URL of the app to test.
<b>Test Name</b>	Required. A name for this load test, used to identify it for reporting and for comparison with other test runs.
<b>User Load</b>	Required. The number of concurrent users to simulate in this test. Select a value from the drop-down list.
<b>Run Duration (sec)</b>	Required. The duration of this test in seconds. Select a value from the drop-down list.
<b>Load Location</b>	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 VSTS account.

ARGUMENT	DESCRIPTION
<b>Run load test using</b>	<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 VSTS account when using this option. See <a href="#">Testing private/intranet applications using Cloud-based load testing</a>.</p>
<b>Fail test if Avg. Response Time (ms) exceeds</b>	<p>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.</p>
<b>Control options</b>	<p>See <a href="#">Control options</a></p>

## More Information

- [Cloud-based Load Testing](#)
- [Performance testing video and Q&A](#)

## Related tasks

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

## Q&A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- Report problems through the [Developer Community](#)
- Send suggestions on [UserVoice](#)

4 min to read •

# Test: Publish Code Coverage Results

2/21/2018 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

{ }  Publishes the code coverage results of a build.

## Demands

The build agent must have the following capabilities:

- MSBuild
- Azure PowerShell

## Arguments

ARGUMENT	DESCRIPTION
<b>Code Coverage Tool</b>	Required. The tool with which code coverage results are generated. Currently supported schemas are <a href="#">Cobertura</a> and <a href="#">Jacoco</a> .
<b>Summary File</b>	Required. The path of the summary file containing code coverage statistics such as line, method, and class coverage. The value may contain <a href="#">minimatch patterns</a> . Example: <code>\$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura/coverage</code>
<b>Report Directory</b>	The path of the code coverage HTML report directory. The report directory is published for subsequent viewing as an artifact of the build. The value may contain <a href="#">minimatch patterns</a> . Example: <code>\$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura</code>
<b>Additional Files</b>	The file path pattern specifying any additional code coverage files to be published as artifacts of the build. The value may contain <a href="#">minimatch patterns</a> . Example: <code>\$(System.DefaultWorkingDirectory)/**/*.*.exec</code>
<b>Control options</b>	See <a href="#">Control options</a>

## More Information

- [Continuous testing scenarios and capabilities](#)

## Related tasks

- [Visual Studio Test](#)
- [Publish Test Results](#)

## Q&A

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- Report problems through the [Developer Community](#)
- Send suggestions on [UserVoice](#)

# Test: Publish Test Results

2/21/2018 • 2 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017 | TFS 2015



Publishes the test results to TFS or VSTS when tests are run using a runner of your choice.

The task supports popular test result formats including [JUnit](#), [NUnit 2](#), [NUnit 3](#), Visual Studio Test (TRX), and [xUnit 2](#). If you use the built-in tasks such as [Visual Studio Test](#) to run tests, results are automatically published and you do not need a separate publish test results task.

## Demands

The build agent must have the following capabilities:

- MSBuild
- Azure PowerShell

## Arguments

ARGUMENT	DESCRIPTION
<b>Test result format</b>	Specify the format of the results files you want to publish. The following formats are supported. - Version 1 of the task: <a href="#">JUnit</a> , <a href="#">xUnit 2</a> , <a href="#">NUnit 2</a> , Visual Studio Test Results (TRX) - Version 2 of the task: <a href="#">JUnit</a> , <a href="#">xUnit 2</a> , <a href="#">NUnit 2</a> , <a href="#">NUnit 3</a> , Visual Studio Test Results (TRX)
<b>Test results files</b>	Use this to specify one or more test results files. - Version 1 of the task: 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. Multiple paths can be specified, separated by a semicolon. - Version 2 of the task: Uses the <a href="#">minimatch patterns</a> . For example, <code>**/TEST-*.xml</code> searches for all the XML files whose names start with <code>TEST-</code> in all subdirectories.
<b>Search folder</b>	Available only in version 2 of the task. Folder to search for the test result files. Default is <code>\$(System.DefaultWorkingDirectory)</code>
<b>Merge test results</b>	When this option is selected, test results from all the files will be reported against a single test run. If this option is not selected, a separate test run will be created for each test result file.
<b>Test run title</b>	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 process can be used.

ARGUMENT	DESCRIPTION
<b>Advanced - Platform</b>	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.
<b>Advanced - Configuration</b>	Build configuration against which the Test Run should be reported. For example, Debug or Release. If you have defined a variable for configuration in your build task, use that here.
<b>Advanced - Upload test results files</b>	When selected, the task will upload all the test result files as attachments to the test run.
<b>Control options</b>	See <a href="#">Control options</a>

## More Information

- [Continuous testing scenarios and capabilities](#)

## Related tasks

- [Visual Studio Test](#)
- [Publish Code Coverage Results](#)

## Q&A

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- Report problems through the [Developer Community](#)
- Send suggestions on [UserVoice](#)

4 min to read •

# Test: Xamarin Test Cloud

9/13/2017 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Test mobile apps with Xamarin Test Cloud using Xamarin.UITest

## Demands

None

## 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 (<code>*</code>) and recursive wildcards (<code>**</code>). For example, <code>**/*.apk</code> 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: <code>*.dSYM</code> <b>Note:</b> This argument applies only to iOS apps.
<b>Team API Key</b>	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.
<b>User Email</b>	Email address of your <a href="#">Xamarin Test Cloud account</a> .
<b>Devices</b>	The devices string generated by Xamarin Test Cloud. To get this string: <ul style="list-style-type: none"><li>- Go to your <a href="#">Xamarin Test Cloud account</a>.</li><li>- Choose <b>New Test Run</b>.</li><li>- Step through the wizard.</li><li>- When you reach the final page, copy the ID that follows the <code>--devices</code> option.</li></ul>
<b>Series</b>	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> .
<b>Test Assembly Directory</b>	Relative path to the folder that contains the test assemblies. For example: <code>SolutionName/TestsProjectName/bin/Release</code>
<b>Advanced - Parallelization</b>	Select None, By test fixture, or By test method.

ARGUMENT	DESCRIPTION
<b>Advanced - System Language</b>	Select your language. If it isn't displayed, select Other and then enter its locale below. For example: <code>en_AU</code> .
<b>Advanced - test-cloud.exe Location</b>	Location of test-cloud.exe. In most cases leave this set to the default value.
<b>Advanced - Optional Arguments</b>	(Optional) Arguments passed to test-cloud.exe. See <a href="#">Submitting UITests at the Command Line</a> .
<b>Advanced - Publish results to VSO/TFS</b>	Select if you want to pass the <code>--nunit-xml</code> option to test-cloud.exe so that results from the NUnit xml file are published to TFS or VSTS.
<b>Control options</b>	See <a href="#">Control options</a>

## Example

[Build your Xamarin app](#)

## Q&A

### 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. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Package: CocoaPods

9/13/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)

 Runs CocoaPods `pod install`

CocoaPods is the dependency manager for Swift and Objective-C Cocoa projects.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Working Directory	Working directory. If you leave it blank, the working directory is the root of your repo.

## Q & A

### What other kinds of apps can I build?

[Build your app](#)

### What other kinds of build steps are available?

[Specify your build steps](#)

### 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. This option is not available for GitHub repos.
- TFVC: [Use gated check-in](#).

### How do I modify other parts of my build definition?

- [Specify your build steps](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

### I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent pipelines. See [Concurrent build and release pipelines in VSTS](#).

## **How do I see what has changed in my build definition?**

[View the change history of your build definition](#)

## **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

## **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

## **I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Package: npm

1/9/2018 • 2 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#) | [TFS 2015](#)



Install and publish npm packages

## Install npm packages

### Demands

[npm](#)

### Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select <code>install</code> here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".

### CUSTOM REGISTRIES AND AUTHENTICATION

Registries to use	<p><i>Leave this section blank to use packages from npmjs directly.</i></p> <p>Otherwise, select one of these options:</p> <p><b>Registries in my .npmrc:</b></p> <ul style="list-style-type: none"><li>Select this option to use feeds specified in a <a href="#">.npmrc</a> file you've checked into source control.</li><li>Credentials for registries outside this account/collection can be used to inject credentials you've provided as an <a href="#">npm service endpoint</a> into your .npmrc as the build runs.</li></ul> <p><b>Use packages from this VSTS/TFS registry:</b></p> <ul style="list-style-type: none"><li>Select this option to use one Package Management feed in the same account/collection as the build.</li></ul>
-------------------	---

### ADVANCED

Verbose logging	Enables verbose logging.
-----------------	--------------------------

### CONTROL OPTIONS

## Publish npm packages

### Demands

[npm](#)

### Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select <code>publish</code> here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
<b>DESTINATION REGISTRY AND AUTHENTICATION</b>	
Registry location	<ul style="list-style-type: none"> <li>• <b>Registry I select here</b> publishes to a Package Management registry in the same account/collection as the build. After you select this option, select the target registry from the dropdown.</li> <li>• <b>External npm registry (including other accounts/collections)</b> publishes to an external server such as <a href="#">npm</a>, <a href="#">MyGet</a>, or a Package Management feed in another VSTS account or TFS collection. After you select this option, create and select an <a href="#">npm service endpoint</a>.</li> </ul>
<b>ADVANCED</b>	
Verbose logging	Enables verbose logging.
<b>CONTROL OPTIONS</b>	

## Custom npm command

### Demands

[npm](#)

### Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select <code>custom</code> here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
Command and arguments	<p>The custom command and arguments you wish to be executed.</p> <p>If your arguments contain double quotes (""), escape them with a slash (\), and surround the escaped string with double quotes ("").</p> <p>Example: to run  <code>npm run myTask -- --users='{"foo":"bar"}'</code>, provide this input: <code>run myTask -- --users="\"foo\";\"bar\""</code>.</p>
<b>CUSTOM REGISTRIES AND AUTHENTICATION</b>	

ARGUMENT	DESCRIPTION
Registries to use	<p><i>Leave this section blank to use packages from npmjs directly.</i></p> <p>Otherwise, select one of these options:</p> <p><b>Registries in my .npmrc:</b></p> <ul style="list-style-type: none"> <li>• Select this option to use feeds specified in a <a href="#">.npmrc</a> file you've checked into source control.</li> <li>• Credentials for registries outside this account/collection can be used to inject credentials you've provided as an <a href="#">npm service endpoint</a> into your <a href="#">.npmrc</a> as the build runs.</li> </ul> <p><b>Use packages from this VSTS/TFS registry:</b></p> <ul style="list-style-type: none"> <li>• Select this option to use one Package Management feed in the same account/collection as the build.</li> </ul>

#### CONTROL OPTIONS

## Examples

[Build: Gulp](#)

[Build your Node.js app with Gulp](#)

## Q&A

**Where can I learn npm commands and arguments?**

[npm docs](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Package: NuGet

1/2/2018 • 9 min to read • [Edit Online](#)

**Version 2.\*** | [Other versions](#)

**VSTS | TFS 2018**



Install and update NuGet package dependencies, or package and publish NuGet packages.

If your code depends on NuGet packages, make sure to add this step before your [Visual Studio Build step](#). Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

## TIP

Looking for help to get started? See the how-to's for [restoring](#) and [publishing](#) packages.

## TIP

This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the [Tool Installer](#).

## NOTE

Using or creating .NET Core or .NET Standard packages? Use the [.NET Core](#) task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

## Restore NuGet packages

### Demandes

None

### Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the <b>Solution</b> argument in your <a href="#">Visual Studio Build step</a> and paste it here, or create a link using the Link button in the information panel.

### FEEDS AND AUTHENTICATION

Feeds to use	<p><b>Feed(s) I select here:</b></p> <ul style="list-style-type: none"><li>Select this option to use NuGet.org and/or one Package Management feed in the same account/collection as the build.</li></ul> <p><b>Feeds in my NuGet.config:</b></p> <ul style="list-style-type: none"><li>Select this option to use feeds specified in a <a href="#">NuGet.config</a> file you've checked into source control.</li><li>Credentials for feeds outside this account/collection can be used to inject credentials you've provided as a <a href="#">NuGet service endpoint</a> into your NuGet.config as the build runs.</li></ul>
<b>ADVANCED</b>	

ARGUMENT	DESCRIPTION
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
<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 steps

 <b>Package: NuGet</b>	Install your NuGet package dependencies. <ul style="list-style-type: none"> <li>Path to solution, packages.config, or project.json: <code>**/*.sln</code></li> <li>Feeds to use: Feeds in my NuGet.config</li> <li>Path to NuGet.config: <code>ConsoleApplication1/NuGet.config</code></li> </ul>
 <b>Build: Visual Studio Build</b>	Build your solution. <ul style="list-style-type: none"> <li>Solution: <code>**\*.sln</code></li> <li>Restore NuGet Packages: <b>(Important)</b> Make sure this option is cleared.</li> </ul>

## Pack NuGet packages

### Demands

None

### Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>**\*.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none"> <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 <a href="#">file matching patterns</a>. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a <a href="#">variable</a> such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
<b>PACK OPTIONS</b>	
Automatic package versioning	<a href="#">This blog post</a> provides an overview of the automatic package versioning available here.
<b>ADVANCED</b>	
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.
Verbosity	Specifies the amount of detail displayed in the output.
<b>CONTROL OPTIONS</b>	

## Push NuGet packages

### Demand

None

### Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none"> <li>Default value: <code>\$(Build.ArtifactStagingDirectory)/*.nupkg</code></li> <li>To publish a single package, click the ... button and select the file.</li> <li>Use <a href="#">file matching patterns</a> to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</li> <li>Use <a href="#">variables</a> to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the <b>package folder</b> in the pack step above, you could specify <code>\$(Build.ArtifactStagingDirectory)\**\*.nupkg</code> here.</li> </ul>
Target feed location	<ul style="list-style-type: none"> <li><b>This account/collection</b> publishes to a Package Management feed in the same account/collection as the build. After you select this option, select the target feed from the dropdown.           <ul style="list-style-type: none"> <li>"Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. This option is currently only available on VSTS.</li> </ul> </li> <li><b>External NuGet server (including other accounts/collections)</b> publishes to an external server such as <a href="#">NuGet</a>, <a href="#">MyGet</a>, or a Package Management feed in another VSTS account or TFS collection. After you select this option, you create and select a <a href="#">NuGet service endpoint</a>.</li> </ul>
<b>ADVANCED</b>	
Verbosity	Specifies the amount of detail displayed in the output.
<b>CONTROL OPTIONS</b>	

## Publishing symbols

When you push packages to a Package Management feed, you can also publish symbols using the [Index Sources & Publish Symbols task](#).

## Publishing packages to TFS with IIS Basic authentication enabled

This task is unable to publish NuGet packages to a TFS Package Management feed that is running on a TFS server with IIS Basic authentication enabled. [See here](#) for more details.

## Custom NuGet command

### Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

## End-to-end example

You want to package and publish some projects in a C# class library to your VSTS 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>

### Option 1: publish to VSTS

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [create a feed](#).
3. Add the following build steps:

 <b>Package: NuGet</b>	Install your NuGet package dependencies. <ul style="list-style-type: none"><li>• Path to solution, packages.config, or project.json: <code>**\*.sln</code></li><li>• Feeds to use: Feeds I select here</li><li>• Use packages from NuGet.org: Checked</li></ul>
 <b>Build: Visual Studio Build</b>	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>



### Package: NuGet

Package your projects.

- Command: pack
- Path to csproj or nuspec file(s) to pack: `**/*.csproj`
- Configuration to Package: `Release`
- Package Folder: `$(Build.ArtifactStagingDirectory)`
- Pack options > Automatic package versioning: Use the build number



### Package: NuGet

Publish your packages to VSTS.

- Command: push
- Path to NuGet package(s) to publish: `$(Build.ArtifactStagingDirectory)`
- Target feed location: This account/collection
- Target feed: Select your feed

### Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [register with NuGet.org](#).
3. Use the steps in the previous section, but substitute the final step for the step shown here.



### Package: NuGet

Publish your packages to NuGet.org.

- Command: push
- Path to NuGet package(s) to publish: `$(Build.ArtifactStagingDirectory)`
- Target feed location: External NuGet server
- NuGet server: Create a new [NuGet service endpoint](#) with your NuGet.org ApiKey and select it here

## Task versions

### Task: NuGet (formerly NuGet Restore at 1.\* , NuGet Installer at 0.\* )

TASK VERSION	VSTS	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018
0.*	Deprecated but available	Appeared in 2017, deprecated in 2017 Update 2

### Task: NuGet Packager

TASK VERSION	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

### Task: NuGet Publisher

TASK VERSION	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

## Task: NuGet Command

TASK VERSION	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2017 Update 2, deprecated in TFS >= 2018

# Q & A

## Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add a VSTS 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 VSTS package management?

[Package Management in VSTS and TFS](#)

## Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

## What other kinds of apps can I build?

[Build your app](#)

## What other kinds of build steps are available?

[Specify your build steps](#)

## 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. This option is not available for GitHub repos.
- TFVC: [Use gated check-in](#).

## How do I modify other parts of my build definition?

- [Specify your build steps](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

## I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent pipelines. See [Concurrent build and release pipelines in VSTS](#).

**How do I see what has changed in my build definition?**

[View the change history of your build definition](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Package: Xamarin component restore

9/13/2017 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#)



Restores [Xamarin components](#) for the specified solution

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Path to Solution	Click the ... button and select your solution.
Email	Xamarin account email address.
Password	Xamarin account password. <b>Important:</b> Use a <a href="#">secret variable</a> to avoid exposing this value.
CONTROL OPTIONS	

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Package: NuGet Installer version 0.\*

1/2/2018 • 1 min to read • [Edit Online](#)

**VSTS (deprecated) | TFS 2017 (deprecated in 2017 Update 2)**



Install and update NuGet package dependencies.

## Demands

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 NuGetPackages** checkbox in that step.

## Arguments

ARGUMENT	DESCRIPTION
Path to Solution	Copy the value from the <b>Solution</b> argument in your <a href="#">Visual Studio Build step</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> .
ADVANCED	
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

### Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```
`-- ConsoleApplication1
  |-- ConsoleApplication1.sln
  |-- NuGet.config
  '-- ConsoleApplication1
    |-- ConsoleApplication1.csproj
```

### Build steps



### Package: NuGet 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.

# Package: NuGet

1/2/2018 • 9 min to read • [Edit Online](#)

**Version 2.\*** | [Other versions](#)

**VSTS | TFS 2018**



Install and update NuGet package dependencies, or package and publish NuGet packages.

If your code depends on NuGet packages, make sure to add this step before your [Visual Studio Build step](#). Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

## TIP

Looking for help to get started? See the how-to's for [restoring](#) and [publishing](#) packages.

## TIP

This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the [Tool Installer](#).

## NOTE

Using or creating .NET Core or .NET Standard packages? Use the [.NET Core](#) task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

## Restore NuGet packages

### Demandes

None

### Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the <b>Solution</b> argument in your <a href="#">Visual Studio Build step</a> and paste it here, or create a link using the Link button in the information panel.

### FEEDS AND AUTHENTICATION

Feeds to use	<b>Feed(s) I select here:</b> <ul style="list-style-type: none"><li>Select this option to use NuGet.org and/or one Package Management feed in the same account/collection as the build.</li></ul> <b>Feeds in my NuGet.config:</b> <ul style="list-style-type: none"><li>Select this option to use feeds specified in a <a href="#">NuGet.config</a> file you've checked into source control.</li><li>Credentials for feeds outside this account/collection can be used to inject credentials you've provided as a <a href="#">NuGet service endpoint</a> into your NuGet.config as the build runs.</li></ul>
<b>ADVANCED</b>	

ARGUMENT	DESCRIPTION
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
<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 steps

 <b>Package: NuGet</b>	Install your NuGet package dependencies. <ul style="list-style-type: none"> <li>Path to solution, packages.config, or project.json: <code>**/*.sln</code></li> <li>Feeds to use: Feeds in my NuGet.config</li> <li>Path to NuGet.config: <code>ConsoleApplication1/NuGet.config</code></li> </ul>
 <b>Build: Visual Studio Build</b>	Build your solution. <ul style="list-style-type: none"> <li>Solution: <code>**\*.sln</code></li> <li>Restore NuGet Packages: <b>(Important)</b> Make sure this option is cleared.</li> </ul>

## Pack NuGet packages

### Demands

None

### Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>**\*.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none"> <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 <a href="#">file matching patterns</a>. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a <a href="#">variable</a> such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
<b>PACK OPTIONS</b>	
Automatic package versioning	<a href="#">This blog post</a> provides an overview of the automatic package versioning available here.
<b>ADVANCED</b>	
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.
Verbosity	Specifies the amount of detail displayed in the output.
<b>CONTROL OPTIONS</b>	

## Push NuGet packages

### Demand

None

### Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none"> <li>Default value: <code>\$(Build.ArtifactStagingDirectory)/*.nupkg</code></li> <li>To publish a single package, click the ... button and select the file.</li> <li>Use <a href="#">file matching patterns</a> to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</li> <li>Use <a href="#">variables</a> to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the <b>package folder</b> in the pack step above, you could specify <code>\$(Build.ArtifactStagingDirectory)\**\*.nupkg</code> here.</li> </ul>
Target feed location	<ul style="list-style-type: none"> <li><b>This account/collection</b> publishes to a Package Management feed in the same account/collection as the build. After you select this option, select the target feed from the dropdown.           <ul style="list-style-type: none"> <li>"Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. This option is currently only available on VSTS.</li> </ul> </li> <li><b>External NuGet server (including other accounts/collections)</b> publishes to an external server such as <a href="#">NuGet</a>, <a href="#">MyGet</a>, or a Package Management feed in another VSTS account or TFS collection. After you select this option, you create and select a <a href="#">NuGet service endpoint</a>.</li> </ul>
<b>ADVANCED</b>	
Verbosity	Specifies the amount of detail displayed in the output.
<b>CONTROL OPTIONS</b>	

## Publishing symbols

When you push packages to a Package Management feed, you can also publish symbols using the [Index Sources & Publish Symbols task](#).

## Publishing packages to TFS with IIS Basic authentication enabled

This task is unable to publish NuGet packages to a TFS Package Management feed that is running on a TFS server with IIS Basic authentication enabled. [See here](#) for more details.

## Custom NuGet command

### Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

## End-to-end example

You want to package and publish some projects in a C# class library to your VSTS 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>

### Option 1: publish to VSTS

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [create a feed](#).
3. Add the following build steps:

 <b>Package: NuGet</b>	Install your NuGet package dependencies. <ul style="list-style-type: none"><li>• Path to solution, packages.config, or project.json: <code>**\*.sln</code></li><li>• Feeds to use: Feeds I select here</li><li>• Use packages from NuGet.org: Checked</li></ul>
 <b>Build: Visual Studio Build</b>	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>



### Package: NuGet

Package your projects.

- Command: pack
- Path to csproj or nuspec file(s) to pack: `**/*.csproj`
- Configuration to Package: `Release`
- Package Folder: `$(Build.ArtifactStagingDirectory)`
- Pack options > Automatic package versioning: Use the build number



### Package: NuGet

Publish your packages to VSTS.

- Command: push
- Path to NuGet package(s) to publish: `$(Build.ArtifactStagingDirectory)`
- Target feed location: This account/collection
- Target feed: Select your feed

### Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [register with NuGet.org](#).
3. Use the steps in the previous section, but substitute the final step for the step shown here.



### Package: NuGet

Publish your packages to NuGet.org.

- Command: push
- Path to NuGet package(s) to publish: `$(Build.ArtifactStagingDirectory)`
- Target feed location: External NuGet server
- NuGet server: Create a new [NuGet service endpoint](#) with your NuGet.org ApiKey and select it here

## Task versions

### Task: NuGet (formerly NuGet Restore at 1.\* , NuGet Installer at 0.\* )

TASK VERSION	VSTS	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018
0.*	Deprecated but available	Appeared in 2017, deprecated in 2017 Update 2

### Task: NuGet Packager

TASK VERSION	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

### Task: NuGet Publisher

Task Version	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

## Task: NuGet Command

Task Version	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2017 Update 2, deprecated in TFS >= 2018

# Q & A

## Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add a VSTS 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 VSTS package management?

[Package Management in VSTS and TFS](#)

## Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

## What other kinds of apps can I build?

[Build your app](#)

## What other kinds of build steps are available?

[Specify your build steps](#)

## 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. This option is not available for GitHub repos.
- TFVC: [Use gated check-in](#).

## How do I modify other parts of my build definition?

- [Specify your build steps](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

## I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent pipelines. See [Concurrent build and release pipelines in VSTS](#).

**How do I see what has changed in my build definition?**

[View the change history of your build definition](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Package: NuGet Packager version 0.\*

1/2/2018 • 4 min to read • [Edit Online](#)

VSTS (deprecated) | TFS 2017 Update 2 and below (deprecated in TFS 2018)



Create a NuGet package from either a .csproj or .nuspec file.

## Demands

None

## Arguments

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 <a href="#">.nuspec file</a>. 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 <a href="#">.nuspec file</a> 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 <a href="#">.nuspec file</a>.</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>

ARGUMENT	DESCRIPTION
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="#">definition options</a>, set the <b>build number format</b> to something like <code>\$(BuildDefinitionName)_\$(Year:yyyy).\$(Month).\$(DayOfMonth)\$(Rev::)</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 Visual Studio Team Services 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>
<b>ADVANCED</b>	
Configuration to Package	<p>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></p>
Additional build properties	<p>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 <a href="#">.nuspec file</a> 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.</p>
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> .
<b>CONTROL OPTIONS</b>	

## Examples

You want to package and publish some projects in a C# class library to your VSTS 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:.rrr)</code>

## Publish to VSTS

Make sure you've prepared the build as described [above](#).

### Create the feed

See [Create a feed](#).

### Build steps

 <b>Build: Visual Studio Build</b>	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>
 <b>Package: NuGet Packager</b>	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>
 <b>Package: NuGet Publisher</b>	Publish your packages to VSTS. <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 steps

 <b>Build: Visual Studio Build</b>	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>
 <b>Package: NuGet Packager</b>	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>
 <b>Package: NuGet Publisher</b>	Publish your packages to NuGet.org. <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:  </li></ul> <ol style="list-style-type: none"><li>1. Click New Service Endpoint, 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: <code>https://nuget.org/</code></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>

# Package: NuGet Publisher version 0.\*

1/2/2018 • 2 min to read • [Edit Online](#)

VSTS (deprecated) | TFS 2017 Update 2 and below (deprecated in TFS 2018)



Publish your NuGet package to a server and update your feed.

## Demands

None

## 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 (<code>*</code>) and recursive wildcards (<code>**</code>) 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 step, 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 Visual Studio Team Services feed. After you select this option, you specify the <a href="#">internal feed URL</a>.</li></ul>
ADVANCED	
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> .
CONTROL OPTIONS	

## Examples

You want to package and publish some projects in a C# class library to your VSTS 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:.rrr)</code>

## Publish to VSTS

Make sure you've prepared the build as described [above](#).

### Create the feed

See [Create a feed](#).

### Build steps

 <b>Build: Visual Studio Build</b>	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>
 <b>Package: NuGet Packager</b>	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>
 <b>Package: NuGet Publisher</b>	Publish your packages to VSTS. <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 steps

 <b>Build: Visual Studio Build</b>	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>
 <b>Package: NuGet Packager</b>	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>
 <b>Package: NuGet Publisher</b>	Publish your packages to NuGet.org. <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:  </li></ul> <ol style="list-style-type: none"><li>1. Click New Service Endpoint, 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: <code>https://nuget.org/</code></li><li>◦ User name: <code>{your-name}</code></li><li>◦ Password TokenName Key: Paste API Key from your <a href="#">NuGet account</a>.</li></ul></li></ol>

# Package: NuGet

1/2/2018 • 9 min to read • [Edit Online](#)

**Version 2.\*** | [Other versions](#)

**VSTS | TFS 2018**



Install and update NuGet package dependencies, or package and publish NuGet packages.

If your code depends on NuGet packages, make sure to add this step before your [Visual Studio Build step](#). Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

**TIP**

Looking for help to get started? See the how-to's for [restoring](#) and [publishing](#) packages.

**TIP**

This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the [Tool Installer](#).

**NOTE**

Using or creating .NET Core or .NET Standard packages? Use the [.NET Core](#) task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

## Restore NuGet packages

### Demands

None

### Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the <b>Solution</b> argument in your <a href="#">Visual Studio Build step</a> and paste it here, or create a link using the Link button in the information panel.

### FEEDS AND AUTHENTICATION

Feeds to use	<p><b>Feed(s) I select here:</b></p> <ul style="list-style-type: none"><li>Select this option to use NuGet.org and/or one Package Management feed in the same account/collection as the build.</li></ul> <p><b>Feeds in my NuGet.config:</b></p> <ul style="list-style-type: none"><li>Select this option to use feeds specified in a <a href="#">NuGet.config</a> file you've checked into source control.</li><li>Credentials for feeds outside this account/collection can be used to inject credentials you've provided as a <a href="#">NuGet service endpoint</a> into your NuGet.config as the build runs.</li></ul>
--------------	---

### ADVANCED

ARGUMENT	DESCRIPTION
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
<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 steps

 <b>Package: NuGet</b>	Install your NuGet package dependencies. <ul style="list-style-type: none"> <li>Path to solution, packages.config, or project.json: <code>**/*.sln</code></li> <li>Feeds to use: Feeds in my NuGet.config</li> <li>Path to NuGet.config: <code>ConsoleApplication1/NuGet.config</code></li> </ul>
 <b>Build: Visual Studio Build</b>	Build your solution. <ul style="list-style-type: none"> <li>Solution: <code>**\*.sln</code></li> <li>Restore NuGet Packages: <b>(Important)</b> Make sure this option is cleared.</li> </ul>

## Pack NuGet packages

### Demands

None

### Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	<p>Specify .csproj files (for example, <code>**\*.csproj</code>) for simple projects. In this case:</p> <ul style="list-style-type: none"> <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 <a href="#">file matching patterns</a>. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</p>
Configuration to package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: <code>Release</code> or <code>\$(BuildConfiguration)</code>
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a <a href="#">variable</a> such as <code>\$(Build.ArtifactStagingDirectory)</code> . If you leave it empty, the package will be created in the root of your source tree.
<b>PACK OPTIONS</b>	
Automatic package versioning	<a href="#">This blog post</a> provides an overview of the automatic package versioning available here.
<b>ADVANCED</b>	
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.
Verbosity	Specifies the amount of detail displayed in the output.
<b>CONTROL OPTIONS</b>	

## Push NuGet packages

### Demands

None

### Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	<p>Specify the packages you want to publish.</p> <ul style="list-style-type: none"> <li>• Default value: <code>\$(Build.ArtifactStagingDirectory)/*.nupkg</code></li> <li>• To publish a single package, click the ... button and select the file.</li> <li>• Use <a href="#">file matching patterns</a> to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains <code>-:</code>, use <code>!</code> instead.</li> <li>• Use <a href="#">variables</a> to specify directories. For example, if you specified <code>\$(Build.ArtifactStagingDirectory)\</code> as the <b>package folder</b> in the pack step above, you could specify <code>\$(Build.ArtifactStagingDirectory)\**\*.nupkg</code> here.</li> </ul>
Target feed location	<ul style="list-style-type: none"> <li>• <b>This account/collection</b> publishes to a Package Management feed in the same account/collection as the build. After you select this option, select the target feed from the dropdown.             <ul style="list-style-type: none"> <li>◦ "Allow duplicates to be skipped" allows you to continually publish a set of packages and only change the version number of the subset of packages that changed. It allows the task to report success even if some of your packages are rejected with 409 Conflict errors. This option is currently only available on VSTS.</li> </ul> </li> <li>• <b>External NuGet server (including other accounts/collections)</b> publishes to an external server such as <a href="#">NuGet</a>, <a href="#">MyGet</a>, or a Package Management feed in another VSTS account or TFS collection. After you select this option, you create and select a <a href="#">NuGet service endpoint</a>.</li> </ul>
<b>ADVANCED</b>	
Verbosity	Specifies the amount of detail displayed in the output.
<b>CONTROL OPTIONS</b>	

## Publishing symbols

When you push packages to a Package Management feed, you can also publish symbols using the [Index Sources & Publish Symbols task](#).

## Publishing packages to TFS with IIS Basic authentication enabled

This task is unable to publish NuGet packages to a TFS Package Management feed that is running on a TFS server with IIS Basic authentication enabled. [See here](#) for more details.

## Custom NuGet command

### Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

## End-to-end example

You want to package and publish some projects in a C# class library to your VSTS 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>

### Option 1: publish to VSTS

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [create a feed](#).
3. Add the following build steps:

 <b>Package: NuGet</b>	Install your NuGet package dependencies. <ul style="list-style-type: none"><li>• Path to solution, packages.config, or project.json: <code>**\*.sln</code></li><li>• Feeds to use: Feeds I select here</li><li>• Use packages from NuGet.org: Checked</li></ul>
 <b>Build: Visual Studio Build</b>	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>

 <b>Package: NuGet</b>	<p>Package your projects.</p> <ul style="list-style-type: none"> <li>• Command: pack</li> <li>• Path to csproj or nuspec file(s) to pack: <code>**/*.csproj</code></li> <li>• Configuration to Package: <code>Release</code></li> <li>• Package Folder: <code>\$(Build.ArtifactStagingDirectory)</code></li> <li>• Pack options &gt; Automatic package versioning: Use the build number</li> </ul>
 <b>Package: NuGet</b>	<p>Publish your packages to VSTS.</p> <ul style="list-style-type: none"> <li>• Command: push</li> <li>• Path to NuGet package(s) to publish: <code>\$(Build.ArtifactStagingDirectory)</code></li> <li>• Target feed location: This account/collection</li> <li>• Target feed: Select your feed</li> </ul>

### Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described [above](#).
2. If you haven't already, [register with NuGet.org](#).
3. Use the steps in the previous section, but substitute the final step for the step shown here.

 <b>Package: NuGet</b>	<p>Publish your packages to NuGet.org.</p> <ul style="list-style-type: none"> <li>• Command: push</li> <li>• Path to NuGet package(s) to publish: <code>\$(Build.ArtifactStagingDirectory)</code></li> <li>• Target feed location: External NuGet server</li> <li>• NuGet server: Create a new <a href="#">NuGet service endpoint</a> with your NuGet.org ApiKey and select it here</li> </ul>
--	--

## Task versions

### Task: NuGet (formerly NuGet Restore at 1.\* , NuGet Installer at 0.\* )

TASK VERSION	VSTS	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018
0.*	Deprecated but available	Appeared in 2017, deprecated in 2017 Update 2

### Task: NuGet Packager

TASK VERSION	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

### Task: NuGet Publisher

Task Version	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2018, deprecated in TFS >= 2018

## Task: NuGet Command

Task Version	VSTS	TFS
0.*	Deprecated but available	Available in TFS < 2017 Update 2, deprecated in TFS >= 2018

## Q & A

### Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project—the location of its packages—is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add a VSTS 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 VSTS package management?

[Package Management in VSTS and TFS](#)

### Where can I learn more about NuGet?

[NuGet Docs](#) Overview

[NuGet Create](#) Packaging and publishing

[NuGet Consume](#) Setting up a solution to get dependencies

### What other kinds of apps can I build?

[Build your app](#)

### What other kinds of build steps are available?

[Specify your build steps](#)

### 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. This option is not available for GitHub repos.
- TFVC: [Use gated check-in](#).

### How do I modify other parts of my build definition?

- [Specify your build steps](#) to run tests, scripts, and a wide range of other processes.
- [Specify build options](#) such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- [Specify the repository](#) to pick the source of the build and modify options such as how the agent workspace is cleaned.
- [Set build triggers](#) to modify how your CI builds run and to specify scheduled builds.
- [Specify build retention policies](#) to automatically delete old builds.

### I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent pipelines. See [Concurrent build and release pipelines in VSTS](#).

**How do I see what has changed in my build definition?**

[View the change history of your build definition](#)

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

4 min to read •

4 min to read •

# Deploy: Azure CLI

1/19/2018 • 3 min to read • [Edit Online](#)

## VSTS

 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.

The task is under development. If you encounter problems, or wish to share feedback about the task and features you would like to see, please [contact us](#).

### What's new in Version 1.0

- Supports the new [AZ CLI 2.0](#), which is Python based
- Works with agents on Linux, macOS, and Windows
- To work with [Azure CLI 1.0](#), which is node based, switch to task version 0.0
- Both versions of Azure-CLI can coexist in the same system, but task V1.0 logs into the Python based AZ CLI using the user's subscription, whereas task V0.0 logs into the node based Azure CLI. Therefore, scripts should include only the appropriate corresponding commands.
- Limitations:
  - No support for Classic subscriptions. AZ CLI 2.0 supports only Azure Resource Manager (ARM) subscriptions
  - Currently, Hosted agents do not have AZ CLI installed. You can either install using `npm install -g azure-cli` or use private agents with AZ CLI pre-installed

## Demands

None

## Prerequisites

- A Microsoft Azure subscription
- A service endpoint connection to your Azure account. You can use either:
  - [Azure Classic service endpoint](#)
  - [Azure Resource Manager service endpoint](#)
- Azure CLI installed on the computer(s) that run the build and release agent. See [Install the Azure CLI](#). If an agent is already running on the machine on which the Azure CLI is installed, restart the agent to ensure all the relevant environment variables are updated.

## Arguments

ARGUMENT	DESCRIPTION
<b>Azure Connection Type</b>	Required. Select the type of service endpoint used to define the connection to Azure. Choose <b>Azure Classic</b> or <b>Azure Resource Manager</b> . 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.

ARGUMENT	DESCRIPTION
<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 endpoint</a> configured for the subscription where the target Azure service, virtual machine, or storage account is located.
<b>Azure RM Subscription</b>	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 endpoint</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.
<b>Script Location</b>	Required. The way that the script is provided. Choose <b>Inline Script</b> or <b>Script Path</b> (the default).
<b>Inline Script</b>	Required if you select <b>Inline Script</b> for the <b>Script Location</b> parameter. Type or copy the script code to execute here. You can include <a href="#">default variables</a> , global variables, and environment variables.
<b>Script Path</b>	Required if you select <b>Script Path</b> for the <b>Script Location</b> parameter. The path to a linked artifact that is the <b>.bat</b> , <b>.cmd</b> , or <b>.sh</b> script you want to run. It can be a fully-qualified path, or a valid path relative to the default working directory.
<b>Arguments</b>	Optional. Any arguments you want to pass to the script.
<b>Advanced - Working Directory</b>	Optional. The working directory in which the script will execute. If not specified, this will be the folder containing the script file.
<b>Advanced - Fail on Standard Error</b>	Set this option if you want the build to fail if errors are written to the <b>StandardError</b> stream.
<b>Control options</b>	See <a href="#">Control options</a>

## Related tasks

- [Azure Resource Group Deployment](#)
- [Azure Cloud Service Deployment](#)
- [Azure Web App Deployment](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

4 min to read •

# Deploy: Azure File Copy

1/19/2018 • 9 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017 | TFS 2015.3



The task is used to copy application files and other artifacts that are required in order to install the app; such as PowerShell scripts, PowerShell-DSC modules, and more.

When the target is Azure VMs, the files are first copied to an automatically generated Azure blob container and then downloaded into the VMs. The container is deleted after the files have been successfully copied to the VMs.

The task uses **AzCopy**, the command-line utility built for fast copying of data from and into Azure storage accounts.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the [Azure Resource Group Deployment](#) task. This task has a sample template that can perform the required operations to set up the WinRM HTTPS protocol on the virtual machines, open the 5986 port in the firewall, and install the test certificate.

## Demands

- none

## 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 endpoint 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 endpoint</a> configured for the subscription where the target Azure service, virtual machine, or storage account is located.
<b>Azure RM Subscription</b>	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 endpoint</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.
<b>Destination Type</b>	Required. The type of target destination for the files. Choose <b>Azure Blob</b> or <b>Azure VMs</b> .

ARGUMENT	DESCRIPTION
<b>Classic Storage Account</b>	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.
<b>RM Storage Account</b>	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.
<b>Container Name</b>	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.
<b>Blob Prefix</b>	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.
<b>Cloud Service</b>	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.
<b>Resource Group</b>	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.
<b>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>Filter Criteria</b>	<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.</li> </ul> <p>Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses.</p> <p>Specify tag names for a filter as {TagName}:{Value} Example:  <input type="text" value="Role:DB;OS:Win8.1"/></p>
<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 <b>username</b>, <b>domain\username</b>, <b>machinename\username</b>, and <b>.\<username></username></b> are supported.</li> <li>- UPN formats such as <b>username@domain.com</b> and built-in system accounts such as <b>NT Authority\System</b> 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>

Argument	Description
<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. 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>
<b>Test Certificate</b>	<p>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).</p>
<b>Output - Storage Container URI</b>	<p>Optional. The name of a variable that will be updated with the URI of the storage container into which the files were copied. Use this variable as an input to subsequent task steps.</p>
<b>Output - Storage Container SAS Token</b>	<p>Optional. The name of a variable that will be updated with the Storage Access Security (SAS) token of the storage container into which the files were copied. Use this variable as an input to subsequent task steps. By default, the SAS token expires after 4 hours.</p>
<b>Control options</b>	<p>See <a href="#">Control options</a></p>

## Related tasks

- [Azure Resource Group Deployment](#)
- [Azure Cloud Service Deployment](#)
- [Azure Web App Deployment](#)

## Q&A

### What are the Azure PowerShell prerequisites for using this task?

The task requires Azure PowerShell to be installed on the machine running the automation agent. The recommended version is 1.0.2, but the task will work with version 0.9.8 and higher. You can use the [Azure PowerShell Installer v1.0.2](#) to obtain this.

### What are the WinRM prerequisites for this task?

The task uses Windows Remote Management (WinRM) HTTPS protocol to copy the files from the storage blob container to the Azure VMs. This requires the WinRM HTTPS service to be configured on the VMs, and a suitable certificate installed.

If the VMs have been created without opening the WinRM HTTPS ports, follow these steps:

1. Configure an inbound access rule to allow HTTPS on port 5986 of each VM.
2. Disable [UAC remote restrictions](#).
3. Specify the credentials for the task to access the VMs using an administrator-level login in the simple form **username** without any domain part.
4. Install a certificate on the machine that runs the automation agent.
5. Set the **Test Certificate** parameter of the task if you are using a self-signed certificate.

For more details, see [this blog post](#).

### What type of service endpoint should I choose?

The following table lists the storage accounts and the service endpoint 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 endpoint
Classic	Azure service endpoint with certificate-based or credentials-based authentication using a school or work account

- For Azure classic resources, use an **Azure** service endpoint 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 endpoint type. More details at [Automating Azure Resource Group deployment using a Service Principal](#).
- If you are using an **Azure Resource Manager** service endpoint type, or an **Azure** service endpoint 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 endpoint type with credentials-based authentication does not filter the storage, Resource Group or cloud service, and virtual machine fields.

#### **What happens if my Resource Group contains both Classic and Resource Manager VMs?**

If the specified Resource Group contains both Azure Resource Manager and Classic VMs, the set of VMs that will be targeted depends on the connection type. For certificate-based connections and credentials-based connections, the copy operation will be performed only on Classic VMs. For Service Principal Name based connections, the copy operation will be performed on only Resource Manager VMs.

#### **How do I create a school or work account for use with this task?**

A suitable account can be easily created for use in a service endpoint:

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 endpoint connection. Deployments will be processed using this account.

#### **Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

#### **I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

#### **I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy: Azure Key Vault

1/19/2018 • 3 min to read • [Edit Online](#)

## Overview



This task is used 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 definition. The task is Node-based, and works with agents on Linux, macOS, and Windows.

## Pre-requisites for the task

The task has the following pre-requisites:

- An Azure subscription linked to Team Foundation Server or VSTS using the [Azure Resource Manager service endpoint](#).
- 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 endpoint 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.

**Parameters of the task:**

PARAMETER	DESCRIPTION
<b>Azure Subscription</b>	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>
<b>Key Vault</b>	Required. Select the name of the Azure Key Vault from which the secrets will be downloaded.
<b>Secrets filter</b>	Required. A comma-separated list of secret names to be downloaded. Use the default value <code>*</code> to download all the secrets from the vault.

#### NOTE

Values are retrieved as strings. For example, if there is a secret named **connectionString**, a task variable `connectionString` is created with the latest value of the respective secret fetched from Azure key vault. This variable is then available in subsequent tasks.

If the value fetched from the vault is a certificate (for example, a PFX file), the task variable will contain the contents of the PFX in string format. You can use the following PowerShell code to retrieve the PFX file from the task variable:

```
$kvSecretBytes = [System.Convert]::FromBase64String($(PfxSecret))
$certCollection = New-Object System.Security.Cryptography.X509Certificates.X509Certificate2Collection
$certCollection.Import($kvSecretBytes,$null,
[System.Security.Cryptography.X509Certificates.X509KeyStorageFlags]::Exportable)
```

If the certificate file will be stored locally on the machine, it is good practice to encrypt it with a password:

```
#Get the file created
$password = 'your password'
$protectedCertificateBytes =
$certCollection.Export([System.Security.Cryptography.X509Certificates.X509ContentType]::Pkcs12, $password)
$pfxPath = [Environment]::GetFolderPath("Desktop") + "\MyCert.pfx"
[System.IO.File]::WriteAllBytes($pfxPath, $protectedCertificateBytes)
```

For more details, see [Get started with Azure Key Vault certificates](#).

## Contact Information

Contact [RM\\_Customer\\_Questions@microsoft.com](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.

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available

on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

4 min to read •

4 min to read •

4 min to read •

# Deploy: Copy Files Over SSH

1/19/2018 • 2 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017



Copy files from source folder to 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.

## 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).

## Arguments

ARGUMENT	DESCRIPTION
<b>SSH endpoint</b>	The name of an SSH service endpoint containing connection details for the remote machine. <ul style="list-style-type: none"><li>- The hostname or IP address of the remote machine, the port number, and the user name are required to create an SSH endpoint.</li><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>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. <ul style="list-style-type: none"><li>- Example: <code>**/*.jar \n **/*.war</code> includes all jar and war files (including sub folders) under the source folder.</li><li>- Example: <code>** \n !**/*.xml</code> includes all files (including sub folders) under the source folder but excludes xml files.</li></ul>
<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.

ARGUMENT	DESCRIPTION
<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>

## See also

- [Install SSH Key task](#)
- [SSH task](#)
- Blog post [SSH build task](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

4 min to read •

4 min to read •

# Deploy: PowerShell on Target Machines

1/19/2018 • 4 min to read • [Edit Online](#)

**VSTS | TFS 2018 | TFS 2017 | TFS 2015**



Execute PowerShell scripts on remote machine(s).

This task can run both PowerShell scripts and PowerShell-DSC scripts.

- For PowerShell scripts, the computers must have PowerShell 2.0 or higher installed.
- For PowerShell-DSC scripts, the computers must have [Windows Management Framework 4.0](#) installed. This is installed by default on Windows 8.1, Windows Server 2012 R2, and later.

## Prerequisites

This task uses [Windows Remote Management](#) (WinRM) to access on-premises physical computers or virtual computers that are domain-joined or workgroup-joined.

### To set up WinRM for on-premises physical computers or virtual machines

Follow the steps described in [domain-joined](#)

### To set up WinRM for Microsoft Azure Virtual Machines

Azure Virtual Machines require WinRM to use the HTTPS protocol. You can use a self-signed Test Certificate. In this case, the automation agent will not validate the authenticity of the certificate as being issued by a trusted certification authority.

- Azure Classic Virtual Machines.** When you create a [classic virtual machine](#) from the Azure portal, the virtual machine is already set up for WinRM over HTTPS, with the default port 5986 already opened in the firewall and a self-signed certificate installed on the machine. These virtual machines can be accessed with no further configuration required. Existing Classic virtual machines can be also selected by using the [Azure Resource Group Deployment](#) task.
- Azure Resource Group.** If you have an [Azure Resource Group](#) already defined in the Azure portal, you must configure it to use the WinRM HTTPS protocol. You need to open port 5986 in the firewall, and install a self-signed certificate.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the [Azure Resource Group Deployment](#) task. This task has a checkbox named **Enable Deployment Pre-requisites**. Select this to automatically set up the WinRM HTTPS protocol on the virtual machines, open port 5986 in the firewall, and install a test certificate. The virtual machines are then ready for use in the deployment task.

## Arguments

ARGUMENT	DESCRIPTION
<b>Machines</b>	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:5986,192.168.34:5</code></li><li>- An output variable from a previous task.</li></ul> 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.

ARGUMENT	DESCRIPTION
<b>Admin Login</b>	The username of either a domain or a local administrative account on the target host(s). - Formats such as <b>username</b> , <b>domain\username</b> , <b>machinename\username</b> , and <b>.\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.
<b>Password</b>	The password for the administrative account specified above. Consider using a secret variable global to the build or release definition to hide the password. Example: <code>\$(passwordVariable)</code>
<b>Protocol</b>	The protocol that will be used to connect to the target host, either <b>HTTP</b> or <b>HTTPS</b> .
<b>Test Certificate</b>	If you choose the <b>HTTPS</b> option, set this checkbox to skip validating the authenticity of the machine's certificate by a trusted certification authority.
<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 Release Management. 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 or 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: - 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 FDQNs or IP addresses. 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>

## Q&A

**Do I need an agent?**

You need at least one agent to run your build or release. Get an [agent](#).

**I can't select a default agent queue and I can't queue my build or release. How do I fix this?**

See [Agent pools and queues](#).

**I use TFS on-premises and I don't see some of these features. Why not?**

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy: Service Fabric Application Deployment

1/19/2018 • 1 min to read • [Edit Online](#)

[VSTS](#) | [TFS 2018](#) | [TFS 2017](#)



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.

## 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.

## 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/**/PublishProfileName</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 endpoint 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

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy: Service Fabric Compose Deploy

1/19/2018 • 2 min to read • [Edit Online](#)



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 <https://docs.microsoft.com/azure/service-fabric/service-fabric-docker-compose>.

### Service Fabric

- This task uses a Service Fabric installation to connect and deploy to a Service Fabric cluster.
- [Azure Service Fabric Core SDK](#) on the build agent.

## Arguments

ARGUMENT	DESCRIPTION
<b>Cluster Connection</b>	The Azure Service Fabric service endpoint 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: <code>\$(System.DefaultWorkingDirectory)/**/drop/projectartifacts/**/docker-compose.yml</code> . <b>Note:</b> 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 <code>fabric:/</code> as a prefix. Application Names within a Service Fabric cluster must be unique.
<b>Registry Credentials Source</b>	Specifies how credentials for the Docker container registry will be provided to the deployment task: <b>Azure Resource Manager Endpoint:</b> An Azure Resource Manager service endpoint and Azure subscription to be used to obtain a service principal ID and key for an Azure Container Registry. <b>Container Registry Endpoint:</b> A Docker registry connection endpoint. 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. <b>Username and Password:</b> Username and password to be used. We recommend you encrypt your password using <code>Invoke-ServiceFabricEncryptText</code> (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. <b>None:</b> No registry credentials are provided (used for accessing public container registries).
<b>Deploy Timeout (s)</b>	Timeout in seconds for deploying the application.

ARGUMENT	DESCRIPTION
<b>Remove Timeout (s)</b>	Timeout in seconds for removing an existing application.
<b>Get Status Timeout (s)</b>	Timeout in seconds for getting the status of an existing application.
<b>Control options</b>	See <a href="#">Control options</a>

Also see: [Service Fabric PowerShell Utility](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy: SSH

1/19/2018 • 2 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017



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.

## 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).

## Arguments

ARGUMENT	DESCRIPTION
<b>SSH endpoint</b>	The name of an SSH service endpoint 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 endpoint. - The private key and the passphrase must be specified for authentication. - A password can be used to authenticate to remote Linux machines, but this is not supported for macOS or Windows systems.
<b>Run</b>	Choose to run either shell commands or a shell script on the remote machine.
<b>Commands</b>	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: <code>cd /home/user/myFolder;build</code>
<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>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>

## See also

- [Install SSH Key task](#)
- [Copy Files Over SSH](#)
- Blog post [SSH build task](#)

## Q&A

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

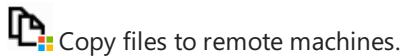
## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

# Deploy: Windows Machine File Copy

2/28/2018 • 2 min to read • [Edit Online](#)

VSTS | TFS 2018 | TFS 2017 | 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.

## Arguments

ARGUMENT	DESCRIPTION
<b>Source</b>	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.
<b>Machines</b>	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>
<b>Admin Login</b>	The username of either a domain or a local administrative account on the target host(s). <ul style="list-style-type: none"><li>- Formats such as <b>domain\username</b>, <b>username</b>, and <b>machine-name\username</b> are supported.</li><li>- UPN formats such as <b>username@domain.com</b> and built-in system accounts such as <b>NT Authority\System</b> are not supported.</li></ul>
<b>Password</b>	The password for the administrative account specified above. Consider using a secret variable global to the build or release definition to hide the password. Example: <code>\$(passwordVariable)</code>
<b>Destination Folder</b>	The folder on the Windows machine(s) to which the files will be copied. Example: <code>C:\FabrikamFibre\Web</code>
<b>Advanced - Clean Target</b>	Set this option to delete all the files in the destination folder before copying the new files to it.
<b>Advanced - Copy Files in Parallel</b>	Set this option to copy files to all the target machines in parallel, which can speed up the copying process.

ARGUMENT	DESCRIPTION
<b>Advanced - Additional Arguments</b>	Arguments to pass to the RoboCopy process. Example: <code>/min:33553332 /1</code>
<b>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>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: - 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 FDQNs or IP addresses. 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>

## Q&A

### I get a system error 53 when using this task. Why?

Typically this occurs when the specified path cannot be located. This may be due to a firewall blocking the necessary ports for file and printer sharing, or an invalid path specification. For more details, see [Error 53](#) on Technet.

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

## Help and support

- See our [troubleshooting](#) page.
- Report any problems on [Developer Community](#), make suggestions on [UserVoice](#), get advice on [Stack Overflow](#), and get support via our [Support](#) page.

4 min to read •

# Tool: Node Tool Installer

10/27/2017 • 1 min to read • [Edit Online](#)

## VSTS

## Build



Finds or downloads and caches the specified version of [Node.js](#) and adds it to the PATH

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
Version Spec	Specify which <a href="#">Node.js version</a> you want to use. Examples: <code>7.x</code> , <code>6.x</code> , <code>6.10.0</code> , <code>&gt;=6.10.0</code>
Check for Latest Version	Select if you want the agent to check for the latest available version that satisfies the version spec. For example, you select this option because you run this build on your <a href="#">private agent</a> and you want to always use the latest <code>6.x</code> version. <b>TIP</b> If you're using <a href="#">our hosted agents</a> , you should leave this check box cleared. We update the hosted agents on a regular basis, but they're often slightly behind the latest version. So selecting this box will result in your build spending a lot of time updating to a newer minor version.
Control options	See <a href="#">Control options</a> .

## Q&A

### Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# Tool: Java Tool Installer

11/17/2017 • 1 min to read • [Edit Online](#)

## VSTS



Acquires a specific version of Java from a user supplied Azure blob, a location in the source or on the agent, or the tools cache and sets JAVA\_HOME. Use this task to change the version of Java used in Java tasks.

## Demands

None

## Arguments

ARGUMENT	DESCRIPTION
JDK Version	Specify which JDK version to download and use.
JDK Architecture	Specify the bit version of the JDK.
JDK source	Specify the source for the compressed JDK, either Azure blob storage or a local directory on the agent or source repository.
JDK file	Applicable when JDK is located in a local directory. Specify the path to the folder that contains the compressed JDK. The path could be in your source repository or a local path on the agent.
Azure Subscription	Applicable when the JDK is located in Azure Blob storage. Specify the Azure Resource Manager subscription for the JDK.
Storage Account Name	Applicable when the JDK is located in Azure Blob storage. Specify the Storage account name in which the JDK is located. Azure Classic and Resource Manager storage accounts are listed.
Container Name	Applicable when the JDK is located in Azure Blob storage. Specify the name of the container in the storage account in which the JDK is located.
Common Virtual Path	Applicable when the JDK is located in Azure Blob storage. Specify the path to the JDK inside the Azure storage container.
Destination directory	Specify the destination directory into which the JDK should be extracted.
Clean destination directory	Select this option to clean the destination directory before the JDK is extracted into it.
Control options	See <a href="#">Control options</a> .

## Q&A

### Where can I learn more about tool installers?

For an explanation of tool installers and examples, see [Tool installers](#).

### Do I need an agent?

You need at least one agent to run your build or release. Get an [agent](#).

### I can't select a default agent queue and I can't queue my build or release. How do I fix this?

See [Agent pools and queues](#).

### I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on [VSTS](#) and not yet available on-premises. Some features are available on-premises if you have [upgraded to the latest version of TFS](#).

# File matching patterns reference

9/12/2017 • 1 min to read • [Edit Online](#)

## Pattern syntax

### Basic patterns

#### Asterisk

`*` matches zero or more characters within a file or directory name. See [examples](#).

#### Question mark

`?` matches any single character within a file or directory name. See [examples](#).

#### Character sets

`[]` matches a set or range of characters within a file or directory name. See [examples](#).

### Double-asterisk

`**` recursive wildcard. For example, `/hello/**/*` matches all descendants of `/hello`.

### Extended globbing

- `?(hello|world)` - matches `hello` or `world` zero or one times
- `*(hello|world)` - zero or more occurrences
- `+(hello|world)` - one or more occurrences
- `@(hello|world)` - exactly once
- `!(hello|world)` - not `hello` or `world`

Note, extended globs cannot span directory separators. For example, `+(/hello/world|other)` is not valid.

### Comments

Patterns that begin with `#` are treated as comments.

### Exclude patterns

Leading `!` changes the meaning of an include pattern to exclude. Interleaved exclude patterns are supported.

Note, multiple leading `!` flips the meaning.

### 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]`.

## 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.tif  
image.png  
image.ico
```

The pattern would match:

```
image.png  
image.ico
```

#### Character set examples

**Example 1:** Given the pattern `Sample[AC].dat` and files:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleD.dat
```

The pattern would match:

```
SampleA.dat  
SampleC.dat
```

**Example 2:** Given the pattern `Sample[A-C].dat` and files:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleD.dat
```

The pattern would match:

```
SampleA.dat  
SampleB.dat  
SampleC.dat
```

**Example 3:** Given the pattern `Sample[A-CEG].dat` and files:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleD.dat  
SampleE.dat  
SampleF.dat  
SampleG.dat  
SampleH.dat
```

The pattern would match:

```
SampleA.dat  
SampleB.dat  
SampleC.dat  
SampleE.dat  
SampleG.dat
```

### Exclude pattern examples

**Example** 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

# File transforms and variable substitution reference

1/19/2018 • 6 min to read • [Edit Online](#)

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.

This configuration 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](#)

## 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 `*.<Environment>.config` and will be executed in the following order:

1. `*.Release.config` (for example, **fabrikam.Release.config**)
2. `*.<Environment>.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 environment 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 definition with an environment named **Release**.
3. Add an **Azure App Service Deploy** task and set (tick) the **XML transformation** option.

4. Save the release definition and start a new release.
5. Open the `Web.config` file to see the transformations from `Web.Release.config`.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="Data Source=(LocalDb)\MSDB;DbFilename=aspcore-local.mdf;" />
    <add name="MyDB"
         connectionString="Data Source=ReleaseSQLServer;Initial Catalog=MyReleaseDB;Integrated
Security=True" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="true" />
  </appSettings>
  <system.web>
    <authentication mode="None" />
    <compilation targetFramework="4.5" />
  </system.web>
</configuration>
```

**Note:**

- You can use this technique to create a default package and deploy it to multiple environments.
- XML transformation takes effect only when the configuration file and transform file are in the same folder within the specified package.
- Set the **Copy to Output Directory** property for the configuration transform files to **Copy If Newer**.
- By default, MSBuild applies the transformation as it generates the web package if the `<DependentUpon>` element is already present in the transform file in the `*.csproj` file. In such cases, the **Azure App Service Deploy** task will fail because there is no further transformation applied on the `Web.config` file. Therefore, it is recommended that the `<DependentUpon>` element is removed from all the transform files to disable any build-time configuration when using XML transformation.

```
...
<Content Include="Web.Debug.config">
    <DependentUpon>Web.config</DependentUpon>
</Content>
<Content Include="Web.Release.config">
    <DependentUpon>Web.config</DependentUpon>
</Content>
...
```

## XML variable substitution

This feature enables you to modify configuration settings in configuration files inside web packages. In this way, the same package can be configured based on the environment to which it will be deployed.

Variable substitution takes effect only on the `applicationSettings`, `appSettings`, `connectionStrings`, and `configSections` elements of configuration files.

### XML variable substitution example

As an example, consider the task of changing the following values in `Web.config`:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSection>
        <section name="entityFramework" />
    </configSection>
    <connectionStrings>
        <!-- Change connectionString in this line: -->
        <add name="DefaultConnection"
            connectionString="Data Source=(LocalDB)\LocalDB;FileName=Local.mdf" />
    </connectionStrings>
    <appSettings>
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobtrusiveJavascriptEnabled" value="true" />
        <!-- Change AdminUserName in this line: -->
        <add key="AdminUserName" value="__AdminUserName__" />
        <!-- Change AdminPassword in this line: -->
        <add key="AdminPassword" value="__AdminPasword__" />
    </appSettings>
    <entityFramework>
        <defaultConnectionFactory type="System.Data.Entity.LocalDbConnectionFactory">
            <parameters></parameters>
        </defaultConnectionFactory>
        <providers>
            <!-- Change invariantName in this line: -->
            <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer" />
        </providers>
    </entityFramework>
</configuration>
```

1. Create a release definition with an environment named **Release**.
2. Add an **Azure App Service Deploy** task and set (tick) the **XML variable substitution** option.

The screenshot shows the 'New Release Definition' page in the Azure DevOps interface. The 'Tasks' tab is active. A 'Deploy Azure App Service' task is selected, indicated by a checkmark icon. In the configuration pane on the right, under 'File Transforms & Variable Substitution Options', the 'XML variable substitution' checkbox is checked and highlighted with a red box.

3. Define the required values in release definition 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 definition and start a new release.

5. Open the `Web.config` file to see the variable substitutions.

```

<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSection>
        <section name="entityFramework" />
    </configSection>
    <connectionStrings>
        <add name="DefaultConnection"
            connectionString="Data Source=(ProdDB)\MSSQLProdDB;AttachFileName=Local.mdf" />
    </connectionStrings>
    <appSettings>
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobtrusiveJavascriptEnabled" value="true" />
        <add key="AdminUserName" value="ProdAdminName" />
        <add key="AdminPassword" value="*password_masked_for_display*" />
    </appSettings>
    <entityFramework>
        <defaultConnectionFactory type="System.Data.Entity.LocalDbConnectionFactory">
            <parameters></parameters>
        </defaultConnectionFactory>
        <providers>
            <provider invariantName="System.Data.SqlClientExtension"
                type="System.Data.Entity.SqlServer" />
        </providers>
    </entityFramework>
</configuration>

```

#### Note:

- By default, ASP.NET applications have a default parameterized connection attribute. These values are overridden only in the `parameters.xml` file inside the web package.
- Because substitution occurs before deployment, the user can override the values in `Web.config` using `parameters.xml` (inside the web package) or a `setparameters` file.

## JSON variable substitution

This feature substitutes values in the JSON configuration files. It overrides the values in the specified JSON configuration files (for example, `appsettings.json`) with the values matching names of release definition and environment variables.

To substitute variables in specific JSON files, provide newline-separated list of JSON files. File names must be specified relative to the root folder. For example, if your package has this structure:

```

/WebPackage(.zip)
  ----- content
    ----- website
      ----- appsettings.json
      ----- web.config
      ----- [other folders]
  --- archive.xml
  --- systeminfo.xml

```

and you want to substitute values in `appsettings.json`, enter the relative path from the root folder; for example `content/website/appsettings.json`. Alternatively, use wildcard patterns to search for specific JSON files. For example, `**/appsettings.json` returns the relative path and name of files named `appsettings.json`.

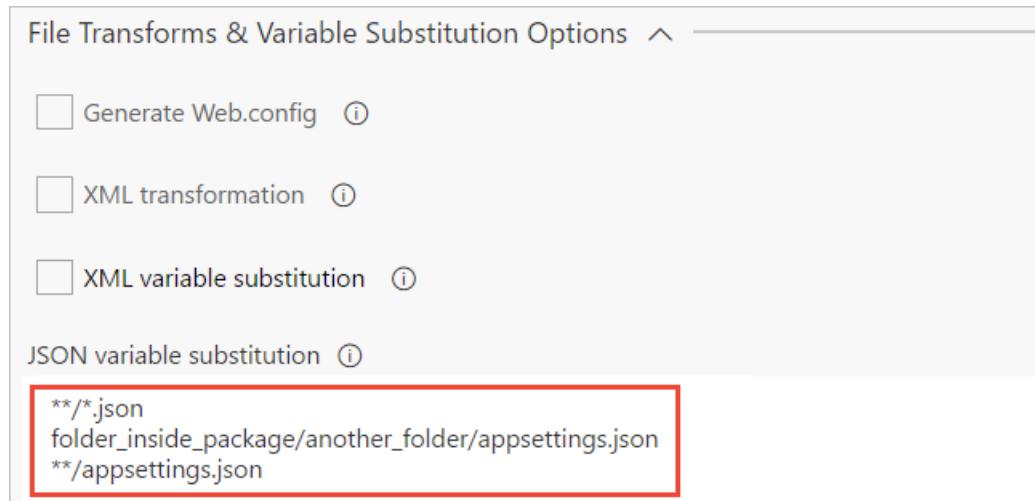
### JSON variable substitution example

As an example, consider the task of overriding values in this JSON file:

```
{
  "Data": {
    "DefaultConnection": {
      "ConnectionString": "Data Source=(LocalDb)\MSDB;AttachDbFilename=aspcore-local.mdf;"
    },
    "DebugMode": "enabled",
    "DBAccess": {
      "Administrators": ["Admin-1", "Admin-2"],
      "Users": ["Vendor-1", "vendor-3"]
    },
    "FeatureFlags": {
      "Preview": [
        {
          "newUI": "AllAccounts"
        },
        {
          "NewWelcomeMessage": "Newusers"
        }
      ]
    }
  }
}
```

The task is to override the values of **ConnectionString**, **DebugMode**, the first of the **Users** values, and **NewWelcomeMessage** at the respective places within the JSON file hierarchy.

1. Create a release definition with an environment 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 definition or environment variables.

NAME	VALUE	SECURE	SCOPE
DebugMode	disabled	No	Release
Data.DefaultConnection.ConnectionString	Data Source=(prodDB)\MSDB;AttachDbFilename=prod.mdf;	No	Release
DBAccess.Users.0	Admin-3	Yes	Release

NAME	VALUE	SECURE	SCOPE
FeatureFlags.Preview.1.NewWelcomeMessage	AllAccounts	No	Release

4. Save the release definition and start a new release.
5. After the transformation, the JSON will contain the following:

```
{
  "Data": {
    "DefaultConnection": {
      "ConnectionString": "Data Source=(prodDB)\MSDB;AttachDbFilename=prod.mdf;"
    },
    "DebugMode": "disabled",
    "DBAccess": {
      "Administrators": ["Admin-1", "Admin-2"],
      "Users": ["Admin-3", "vendor-3"]
    },
    "FeatureFlags": {
      "Preview": [
        {
          "newUI": "AllAccounts"
        },
        {
          "NewWelcomeMessage": "AllAccounts"
        }
      ]
    }
  }
  ...
}
```

**Note:**

- To substitute values in nested levels of the file, concatenate the names with a period ( `.` ) in hierarchical order.
- A JSON object may contain an array whose values can be referenced by their index. For example, to substitute the first value in the **Users** array shown above, use the variable name `DBAccess.Users.0`. To update the value in **NewWelcomeMessage**, use the variable name `FeatureFlags.Preview.1.NewWelcomeMessage`.
- Only **String** substitution is supported for JSON variable substitution.
- Substitution is supported for only UTF-8 and UTF-16 LE encoded files.
- If the file specification you enter does not match any file, the task will fail.
- Variable name matching is case-sensitive.

# Build and release permissions and roles (Security)

2/16/2018 • 9 min to read • [Edit Online](#)

## VSTS | TFS 2018 | TFS 2017 | TFS 2015

To support security of your build and release operations, you can add users to a built-in security group, set individual permissions for a user or group, or add users to pre-defined roles. You manage security for the following objects from the **Build and Release** hub of the web portal, either from the user or admin context.

This topic provides a description of the permissions and roles used to secure operations. To learn how to set permissions or add a user or group to a role, see [Set build and release permissions](#).

For permissions, you grant or restrict permissions by setting the permission state to Allow or Deny, either for a security group or an individual user. For a role, you add a user or group to the role. To learn more about how permissions are set, including inheritance, see [About permissions and groups](#). To learn how inheritance is supported for role-based membership, see [About security roles](#).

## Default permissions assigned to built-in security groups

Once you have been added as a team member, you are a member of the Contributors group. This allows you to define and manage builds and releases. The most common built-in groups include Readers, Contributors, and Project Administrators. These groups are assigned the default permissions as listed below.

TASK	STAKEHOLDERS	READERS	CONTRIBUTORS	BUILD ADMINS	ACCOUNT OWNER/PROJECT ADMINS	RELEASE ADMINS
View build and release definitions		✓	✓	✓	✓	✓
Define builds with continuous integration			✓	✓	✓	
Define releases, manage deployments, manage releases with Release Management			✓		✓	✓
Approve releases	✓		✓		✓	✓
Package Management (5 users free)			✓		✓	✓

Queue builds, edit build quality		✓	✓	✓	
Manage build queues and build qualities			✓	✓	
Manage build retention policies, delete and destroy builds		✓	✓	✓	
Administer build permissions			✓	✓	
Manage release permissions				✓	✓
Create and edit task groups		✓	✓	✓	✓
Manage task group permissions			✓	✓	✓
Can view library items such as variable groups	✓	✓	✓	✓	✓
Use and manage library items such as variable groups			✓	✓	✓

## Security of agents and library entities

You use pre-defined roles and manage membership in those roles to configure [security on agent pools and queues](#). You can configure this in a hierarchical manner either for all pools and queues, or for an individual pool or queue.

Roles are also defined to help you configure security on shared [library entities](#) such as [variable groups](#) and [service endpoints](#). Membership of these roles can be configured hierarchically, as well as at either team project level or individual entity level.

## Build permissions

Permissions in Build follow a hierarchical model. Defaults for all the permissions can be set at the team project level and can be overridden on an individual build definition.

To set the permissions at project level for all build definitions in a project, choose **Security** from the action bar on

the main page of Builds hub.

To set or override the permissions for a specific build definition, choose **Security** from the context menu of the build definition.

The following permissions are defined in Build. All of these can be set at both the levels.

PERMISSION	DESCRIPTION
<b>Administer build permissions</b>	Can change any of the other permissions listed here.
<b>Queue builds</b>	Can queue new builds.
<b>Delete build definition</b>	Can delete build definition(s).
<b>Delete builds</b>	Can delete builds for a definition. Builds that are deleted are <a href="#">retained</a> in the <b>Deleted</b> tab for a period of time before they are destroyed.
<b>Destroy builds</b>	Can delete builds from the <b>Deleted</b> tab.
<b>Edit build definition</b>	Can save any changes to a build definition, including configuration variables, triggers, repositories, and retention policy.
<b>Edit build quality</b>	Can add tags to a build.
<b>Override check-in validation by build</b>	Applies to <a href="#">TFVC gated check-in builds</a> . This does not apply to PR builds.
<b>Retain indefinitely</b>	Can toggle the retain indefinitely flag on a build.
<b>Stop builds</b>	Can stop builds queued by other team members or by the system.
<b>View build definition</b>	Can view build definition(s).
<b>View builds</b>	Can view builds belonging to build definition(s).
<b>Update build information</b>	It is recommended to leave this alone. It's intended to enable service accounts, not team members.
<b>Manage build qualities</b>	<i>Only applies to XAML builds</i>
<b>Manage build queue</b>	<i>Only applies to XAML builds</i>

Default values for all of these permissions are set for team project collections and team 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 definition** and **View releases**.

When it comes to security, there are different best practices and levels of permissiveness. While there's no one right way to handle permissions, we hope these examples help you empower your team to work securely with builds.

- By default, contributors in a team project cannot create or edit build definitions. To grant permissions to work on build definitions, select **Contributors** and set the **Edit build definition** permission to **Allow**.

- In many cases you probably also want to set **Delete build definition** to *Allow*. Otherwise these team members can't delete even their own build definitions.
- Without **Delete builds** permission, users cannot delete even their own completed builds. However, keep in mind that they can automatically delete old unneeded builds using [retention policies](#).
- We recommend that you do not grant these permissions directly to a person. A better practice is to add the person to the build administrator group or another group, and manage permissions on that group.

## Release permissions

Permissions in Release Management follow a hierarchical model. Defaults for all the permissions can be set at the team project level and can be overridden on an individual release definition. Some of the permissions can also be overridden on a specific environment within a definition. The hierarchical model helps you define default permissions for all definitions at one extreme, and to lock down the production environment for an application at the other extreme.

To set permissions at project level for all release definitions in a project, open the shortcut menu from the ▾ icon next to **All release definitions** and choose **Security**.

To set or override the permissions for a specific release definition, open the shortcut menu from the ▾ icon next to that definition name. Then choose **Security** to open the **Permissions** dialog.

To specify security settings for individual environments in a release definition, open the **Permissions** dialog by choosing **Security** on the shortcut menu that opens from the ellipses (...) on an environment in the release definition editor.

The following permissions are defined in Release Management. The scope column explains whether the permission can be set at the team project, release definition, or environment level.

PERMISSION	DESCRIPTION	SCOPES
<b>Administer release permissions</b>	Can change any of the other permissions listed here.	Project, Release definition, Environment
<b>Create releases</b>	Can create new releases.	Project, Release definition
<b>Delete release definition</b>	Can delete release definition(s).	Project, Release definition
<b>Delete release environment</b>	Can delete environment(s) in release definition(s).	Project, Release definition, Environment
<b>Delete releases</b>	Can delete releases for a definition.	Project, Release definition
<b>Edit release definition</b>	Can save any changes to a release definition, including configuration variables, triggers, artifacts, and retention policy as well as configuration within an environment of the release definition. To make changes to a specific environment in a release definition, the user also needs <b>Edit release environment</b> permission.	Project, Release definition

PERMISSION	DESCRIPTION	SCOPES
<b>Edit release environment</b>	Can edit environment(s) in release definition(s). To save the changes to the release definition, the user also needs <b>Edit release definition</b> permission. This permission also controls whether a user can edit the configuration inside the environment of a specific release instance. The user also needs <b>Manage releases</b> permission to save the modified release.	Project, Release definition, Environment
<b>Manage deployments</b>	Can initiate a direct deployment of a release to an environment. This permission is only for direct deployments that are manually initiated by selecting the <b>Deploy</b> or <b>Redeploy</b> actions in a release. If the condition on an environment is set to any type of automatic deployment, the system automatically initiates deployment without checking the permission of the user that created the release.	Project, Release definition, Environment
<b>Manage release approvers</b>	Can add or edit approvers for environment(s) in release definition(s). This permissions also controls whether a user can edit the approvers inside the environment of a specific release instance.	Project, Release definition, Environment
<b>Manage releases</b>	Can edit the configuration in releases. To edit the configuration of a specific environment in a release instance, the user also needs <b>Edit release environment</b> permission.	Project, Release definition
<b>View release definition</b>	Can view release definition(s).	Project, Release definition
<b>View releases</b>	Can view releases belonging to release definition(s).	Project, Release definition

Default values for all of these permissions are set for team project collections and team 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 definition** and **View releases**.

## Task group permissions

Task group permissions follow a hierarchical model. Defaults for all the permissions can be set at the team project level and can be overridden on an individual task group definition.

You use task groups to encapsulate a sequence of tasks already defined in a build or a release definition into a single reusable task. You [define and manage task groups](#) in the **Task groups** tab of the **Build and Release** hub.

PERMISSION	DESCRIPTION
<b>Administer task group permissions</b>	Can add and remove users or groups to task group security.
<b>Delete task group</b>	Can delete a task group.
<b>Edit task group</b>	Can create, modify, or delete a task group.

## Library roles and permissions

Permissions for library artifacts, such as variable groups and secure files, are managed by roles. You use a variable group to store values that you want to make available across multiple build and release definitions. You [define and manage variable groups](#) and [secure files](#) in the **Library** tab of the **Build and Release** hub.

ROLE	DESCRIPTION
<b>Administrator</b>	Can use and manage library items.
<b>Reader</b>	Can only read library items.
<b>User</b>	Can use library items, but not manage them.

## Service endpoint security roles

You [add users to the following roles](#) from the project-level admin context, **Services** page. To create and manage these resources, see [Service endpoints for build and release](#).

ROLE	DESCRIPTION
<b>User</b>	Can use the endpoint when authoring build or release definitions.
<b>Administrator</b>	Can manage membership of all other roles for the service endpoint as well as use the endpoint to author build or release definitions. The system automatically adds the user that created the service endpoint to the Administrator role for that pool.

## Deployment pool security roles

You [add users to the following roles](#) from the collection-level admin context, **Deployment Pools** page. To create and manage deployment pools, see [Deployment groups](#).

ROLE	DESCRIPTION
<b>Reader</b>	Can only view deployment pools.
<b>Service Account</b>	Can view agents, create sessions, and listen for jobs from the agent pool.
<b>User</b>	Can view and use the deployment pool for creating deployment groups.

ROLE	DESCRIPTION
<b>Administrator</b>	Can administer, manage, view and use deployment pools.

## Related notes

- [Set build and release permissions](#)
- [Default permissions and access](#)
- [Permissions and groups reference](#)