Contents

Build and Release

Overview

- About Build and Release

Quickstarts

- Your first build and release
- Build a repo with YAML
- Azure DevOps project

Languages

- .NET
- .NET Core
- Android
- C/C++
- Docker
- Go
- Java
- JavaScript
- Python
- Xamarin
- Xcode

Targets

- Azure Kubernetes Service
- Azure Stack
- Azure SQL database
- Azure Web App
- Linux VM
- npm
- NuGet
- SCVMM
- VMware

- Web App for Containers
- Windows VM
- Tutorials
  - Build multiple branches
  - Build your GitHub repo
  - Build OSS repos
  - CD to Azure web app
  - Set up multi-stage release
  - Use approvals and gates
  - Jenkins CI for VSTS Azure CD
  - Jenkins CI for VSTS Azure AKS
- Concepts
  - Agents
    - Build and release agents
    - Agent pools and queues
    - Microsoft-hosted agents
    - Self-hosted Windows agents
    - Windows agents (TFS 2015)
    - Self-hosted macOS agents
    - Self-hosted Linux agents
    - Running behind a proxy
    - Using self-signed certificate
  - Builds
    - Artifacts
    - History
    - Options
    - Repository
    - Triggers
    - Variables
  - Releases
    - What is Release Management?
    - Release pipelines

- Environments
- Deployment groups
- Provision deployment groups
- Artifacts
- Triggers
- Variables
- Approvals and gates
  - Approvals
  - Gates
- Templates
- Releases and deployments
- Process
- Tasks
- Phases
  - Server phases
  - Deployment group phases
  - Multiple phases
- Conditions
- Licensing
- Pipelines in TFS
- Jobs in VSTS
- Library
- Variable groups
- Task groups
- Service connections
- Secure files
- Retention policies
- How-to Guides
- Connect to Azure
- 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
  - Extend IIS Server deployments
  - IIS servers (WinRM)
  - Azure Government Cloud
- Test your apps
  - Unit test with builds
  - UI test with Selenium
  - Test Java applications
  - Review test results
  - Set up environments
  - Set up test builds
  - Use Test Impact Analysis
  - FAQs for testing
- Publish symbols for debugging
  - Publish symbols
- Use packages in CI/CD
  - NuGet
    - Restore from Package Management
    - Set up Jenkins + NuGet
  - Maven
    - Use Package Management
- Migrate
  - Migrate from XAML builds
  - Migrate from Lab Management
  - Use Build & Release for automated testing
- Set build and release permissions (Security)
- C/C++ with GCC
- Maven

- .NET for Windows
- Universal Windows Platform
- Troubleshooting
  - Troubleshoot build and release
  - Debug deployment issues
  - Troubleshoot Azure connections
- Reference
  - Build tasks
    - .NET Core
    - .NET Core CLI
    - Android build
    - Android signing
    - Ant
    - CMake
    - Docker
    - Docker Compose
    - Go
    - Gradle
    - Grunt
    - Gulp
    - Index Sources & Publish Symbols
    - Jenkins Queue Job
    - Maven
    - MSBuild
    - Visual Studio Build
    - Xamarin.Android
    - Xamarin.iOS
    - Xcode
    - Xcode Package iOS
  - Utility tasks
    - Archive files
    - Azure Network Load Balancer

- Bash
- Batch script
- Command line
- Copy and Publish Build Artifacts
- Copy Files
- cURL Upload Files
- Decrypt File
- Delay
- Delete Files
- Download Secure File
- Download Build Artifacts
- Download Package
- Extract Files
- FTP Upload
- Install Apple Certificate
- Install Apple Provisioning Profile
- Install SSH Key
- Jenkins Download Artifacts
- Invoke Azure Function
- Invoke REST API
- Manual Intervention
- PowerShell
- Publish Build Artifacts
- Publish to Azure Service Bus
- Python Script
- Query Azure Monitor Alerts
- Query Work Items
- Service Fabric PowerShell
- Shell script
- Update Service Fabric App Versions

Test tasks

- App Center Test

- Cloud-based Load Test
- Cloud-based Web Performance Test
- Publish Code Coverage Results
- Publish Test Results
- Run Functional Tests
- Run Distributed Tests
- Run Apache JMeter Load Test
- Visual Studio Test
- Visual Studio Test Agent Deployment
- Xamarin Test Cloud

Package tasks

- CocoaPods
- Conda Environment
- npm
- npm Authenticate
- NuGet
- PyPI Publisher
- Xamarin Component Restore
- Previous versions
  - NuGet Installer 0.*
  - NuGet Restore 1.*
  - NuGet Packager 0.*
  - NuGet Publisher 0.*
  - NuGet Command 0.*

Deploy tasks

- App Center Distribute
- Azure App Service Deploy
- Azure App Service Manage
- Azure Cloud PowerShell Deployment
- Azure CLI
- Azure File Copy
- Azure Key Vault

- Azure Monitor Alerts
- Azure MySQL Deployment
- Azure PowerShell
- Azure Resource Group Deployment
- Azure RM Web App Deployment
- Azure VM Scale Set Deployment
- Chef
- Chef Knife
- Copy Files Over SSH
- Docker
- Docker Compose
- Helm Deploy
- IIS Web App Deployment on Machine Group
- IIS Web App Management on Machine Group
- Kubernetes
- Packer Build
- PowerShell on Target Machines
- Service Fabric App Deployment
- Service Fabric Compose Deploy
- SQL Azure DACPAC Deployment
- SQL DACPAC Deployment on Machine Group
- SSH
- Windows Machine File Copy

Tool tasks

- Go Tool Installer
- Helm Installer
- Java Tool Installer
- .NET Core Tool Installer
- Node Tool Installer
- Use Python Version
- Use Ruby Version
- Visual Studio Test Platform Installer

File matching patterns

File and variable transform

Permissions & security roles

NOTE

Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

VSTS and Team Foundation Server help you implement a continuous integration (CI), test, 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
- Python
- 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

- Visual Studio Test
- Selenium Test
- Review results

Step-by-step tutorials

- CI builds for Git in VSTS
- Set up multi-stage release

Videos

https://channel9.msdn.com/Events/Connect/2017/T174/player	https://channel9.msdn.com/Events/Connect/2017/T168/player
https://channel9.msdn.com/Events/Connect/2017/T170/player	https://channel9.msdn.com/Events/Connect/2017/T171/player
https://channel9.msdn.com/Events/Visual-Studio/Visual-Studio-2017-Launch/190/player	

Concepts

- Release pipelines
- 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
- Manual and exploratory testing
- Load and performance testing
- Unit testing

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

7/9/2018 • 3 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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.

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

Version control systems

The starting point for configuring CI and CD for your applications is to have your source code in a version control system. 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 pipeline. A build pipeline 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 pipeline 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.

Continuous testing

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

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

Create your first build and release

7/20/2018 • 14 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

> **NOTE**
>
> This guidance applies to TFS version 2017.3 and newer.

We'll show you how to use VSTS pipelines to create a build that prints "Hello world" and then automatically creates a release that does the same. By the time you finish here, you'll see an end-to-end process run every time you push new code into your project. You'll also become familiar with all the basic concepts of VSTS pipelines.

We'll show you how to use TFS to create a build that prints "Hello world" and then automatically creates a release that does the same. By the time you finish here, you'll see an end-to-end process run every time you push new code into your project. You'll also become familiar with all the basic concepts of build and release in TFS.

Prerequisites

- A VSTS organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- A self-hosted Windows agent.

Add a script to your repository

Create a PowerShell script that prints `Hello world`.

1. Go to the **Code** hub.
2. Add a file.
   - New navigation
   - Previous navigation

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

```
HelloWorld.ps1
```

4. Copy and paste this script.

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

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

> In this tutorial, our focus is on CI/CD, so we're keeping the code part simple. We're working in a VSTS Git repository directly in your web browser.
>
> When you're ready to begin building and deploying a real app, you can use a wide range of version control clients and services with VSTS CI builds. Learn more.

Create a build pipeline

Create a build pipeline that prints "Hello world."

1. Select the **Build and Release** hub in your VSTS project, and then choose **Builds**.

   - New navigation
   - Previous navigation

2. Create a new pipeline.

   - New navigation
   - Previous navigation

3. Make sure that the **source**, **project**, **repository**, and default **branch** match the location in which you created the script.

4. Start with an **empty process**.

5. On the left side, select **Process** and specify whatever **Name** you want to use. For the **Agent queue**, select **Hosted VS2017**.

6. On the left side, select the plus sign **(+)** to add a task to **Phase 1**. On the right side, select the **Utility** category, select the **PowerShell** task from the list, and then choose **Add**.

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

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

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

1. Select the **Build and Release** hub in your VSTS project, and then choose **Builds**.

2. Create a new pipeline.

3. Start with an **empty process**

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

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

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

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

created.

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

1. Select the **Build and Release** hub in your VSTS project, and then the **Builds** tab.

2. Create a new pipeline.

3. Start with an **empty process**.

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

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

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

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

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

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

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

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

Publish an artifact from your build

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

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

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

Artifact name: Enter `drop`.

Artifact publish location: Select **Visual Studio Team Services/TFS**.

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

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

Artifact Name: Enter `drop`.

Artifact Type: Select **Server**.

> Artifacts are the files that you want your build to produce. Artifacts can be nearly anything your team needs to test or deploy your app. For example, you've got a .DLL and .EXE executable files and .PDB symbols file of a C# or C++ .NET Windows app.
>
> To enable you to produce artifacts, we provide tools such as copying with pattern matching, and a staging directory in which you can gather your artifacts before publishing them. See Artifacts in Team Build.

Enable continuous integration (CI)

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

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

Save and queue the build

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

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

   This queues a new build on the Microsoft-hosted agent.
3. You see a link to the new build on the top of the page.

   - New navigation

- Previous navigation

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

4. Go to the build summary.

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

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

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

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

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

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

4. Go to the build summary.

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

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

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

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

3. Go to the build summary.

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

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

Add some variables and commit a change to your script

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

1. Edit your build pipeline.

2. On the **Tasks** tab, select the PowerShell script task.

3. Add these arguments.

Arguments

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

4. Save the build pipeline.
5. Go to the **Code** hub, **Files** tab.
6. Select the **HelloWorld.ps1** file, and then **Edit** the file.
7. Change the script as follows:

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

8. **Commit** (save) the script.
9. Go to the **Build and Release** hub and select **Queued**. Notice under the **Queued or running** section that a build is automatically triggered by the change that you committed.
10. Select the new build that was created and view its log.
11. Notice that the person who changed the code has their name printed in the greeting message. You also see printed that this was a CI build.

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

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

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

Create a release pipeline

Define the process for running the script in two environments.

1. Go to the **Build and Release** tab, and then select **Releases**.
2. Select the action to create a **New pipeline**. If a release pipeline is already created, select the plus sign **(+)** and then select **Create release pipeline**.
3. Select the action to start with an **empty process**.

4. Name the environment **QA**.

5. In the Artifacts panel, select **+ Add** and specify a **Source (Build pipeline)**. Select **Add**.

6. Select the **Lightning bolt** to trigger continuous deployment and then enable the **Continuous deployment trigger** on the right.

7. Select the **Tasks** tab and select your **QA** environment.

8. Select the plus sign **(+)** for the phase to add a task to the phase.

9. On the **Add tasks** dialog box, select **Utility**, locate the **PowerShell** task, and then select its **Add** button.

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

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

12. Add these **Arguments**:

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

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

14. Rename the cloned environment **Production**.

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

16. Save the release pipeline.

1. Go to the **Build and Release** hub, and then to the **Releases** tab.
2. Select the action to create a **New pipeline**.
3. On the dialog box, select the **Empty** template and select **Next**.
4. Make sure that your **Hello world** build pipeline that you created above is selected. Select **Continuous deployment**, and then select **Create**.
5. Select **Add tasks** in the environment.
6. On the **Task catalog** dialog box, select **Utility**, locate the **PowerShell** task, and then select its **Add** button. Select the **Close** button.
7. For the **Script Path** argument, select the **...** button to browse your artifacts and select the script you created.
8. Add these **Arguments**:

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

9. Rename the environment **QA**.

10. **Clone** the **QA** environment.

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

11. Rename the new environment **Production**.

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

13. Save the release pipeline.

> A release pipeline is a collection of environments to which the application build artifacts are deployed. It also defines the actual deployment process for each environment, as well as how the artifacts are promoted from one environment to another.
>
> Also, notice that we used some variables in our script arguments. In this case, we used release variables instead of the build variables we used for the build pipeline.

Deploy a release

Run the script in each environment.

1. Create a new release.

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

3. Open the release that you just created.

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

1. Create a new release.

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

3. Open the release that you just created.

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

1. Create a new release.

2. Open the release that you just created.

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

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

Change your code and watch it automatically deploy to production

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

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

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

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

3. Select the **Builds** tab to see the build queued and run.

4. After the build is completed, select the **Releases** tab, open the new release, and then go to the **Logs**.

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

> In many cases, you probably would want to edit the release process so that the production deployment happens only after some testing and approvals are in place. See Approvals and gates overview.

Next steps

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

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

LANGUAGE	TEMPLATE TO USE
JavaScript	NodeJS with Gulp (Grunt is also an option)
Xcode	Xcode

Q & A

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

What is Continuous Integration?

What is Continuous Delivery?

What is DevOps?

What kinds of version control can I use

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

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

- Clients
  - Visual Studio Code for Windows, macOS, and Linux
  - Visual Studio with Git for Windows or Visual Studio for Mac
  - Visual Studio with TFVC
  - Eclipse
  - Xcode
  - IntelliJ
  - Command line
- Services
  - VSTS
  - Git service providers such as GitHub and Bitbucket
  - Subversion

How do I replicate a pipeline?

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

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

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

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

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

How do I work with drafts?

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

You can edit and test your draft as needed.

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

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

What else can I do when I queue a build?

You can queue builds automatically or manually.

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

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

Where can I learn more about build pipeline settings?

To learn more about build pipeline settings, see:

- Getting sources
- Tasks
- Variables
- Triggers

- Options
- Retention
- History

How do I programmatically create a build pipeline?

REST API Reference: Create a build pipeline

> **NOTE**
>
> You can also manage builds and build pipelines from the command line or scripts using the VSTS CLI.

Build a repository with YAML

7/19/2018 • 4 minutes to read • Edit Online

If you manage your code in GitHub or in Visual Studio Team Services (VSTS) Git, then you can use VSTS pipelines to easily build the repo by adding a YAML file to it. VSTS pipelines can be used to automate continuous integration (CI) for an application written in any programming language.

> **NOTE**
> To use YAML you must have the **Build YAML pipelines** preview feature enabled on your organization.

Prerequisites

- A VSTS organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.

Get the sample code

Choose the sample application for the language of your choice from one of the following repositories.

PROGRAMMING LANGUAGE	REPOSITORY WITH SAMPLE APPLICATION
.NET Core	`https://github.com/adventworks/dotnetcore-sample`
Android	`https://github.com/adventworks/android-sample`
Go	`https://github.com/adventworks/go-sample`
Java	`https://github.com/adventworks/java-sample`
JavaScript	`https://github.com/adventworks/nodejs-sample`
PHP	`https://github.com/adventworks/php-sample`
Python	`https://github.com/adventworks/python-sample`

Get the code for the sample application into your own GitHub or VSTS repository.

- VSTS Git repo
- GitHub repo

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

1. On the **Code** hub for your project in VSTS, 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.

Get your first build

- VSTS Git repo
- GitHub repo

1. In VSTS, navigate to the **Code** hub, choose the **Files** tab, and then choose the repository you created in the above steps.

2. Inspect the `.vsts-ci.yml` file at the root of your imported repository. The YAML file contains the instructions for the build process. Here's an example snippet from a Gradle build pipeline. The actual content in your file depends on the sample application you chose.

```
queue: 'Hosted VS2017'

steps:
- task: Gradle@2
 inputs:
   gradleWrapperFile: 'gradlew'
   testResultsFiles: '**/TEST-*.xml'
   tasks: 'build'
```

   The next time you push a change to this YAML file, VSTS automatically builds your code.

3. Choose **Edit** to make a change to the YAML file.

4. Add the following comment:

```
# This repository is built using VSTS.
```

5. Commit the above change to the master branch.

6. Navigate to the **Build and Release** hub.

7. Observe that there's a new build pipeline named *{name-of-your-repo} YAML CI*. A build is queued; its status could be either not started or running. Choose the number of the build: *{year}{month}{day}.1*.

8. After an agent is assigned to your job and the agent is initialized, then you'll see information about the build in the console.

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 learned the basics of using YAML to create and run a VSTS build process. This pipeline automatically builds and validates whatever code is checked in by your team. Now you're ready to configure your CI pipeline for

the programming language you're using.

- .NET Core
- Android
- Go
- Java
- JavaScript

Create an Azure Devops project

7/19/2018 • 5 minutes to read • Edit Online

The Azure DevOps Project presents a simplified experience to create Azure resources and to set up a continuous integration (CI) and continuous delivery (CD) pipeline for your application to those resources. It uses Visual Studio Team Services (VSTS), which is Microsoft's DevOps solution for Azure, to configure the pipeline.

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

Sign in to the Azure portal

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

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

Select a sample application and Azure service

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

Configure VSTS and an Azure subscription

1. Create a **new** VSTS organization or choose an **existing** organization. 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**.

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 organization, 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 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 organization. Follow the steps below to view the repository and make code changes to your application. Some of these steps will vary depending on the language that you started with.

1. On the left-hand side of the DevOps project dashboard, select the link for your **master** branch. This link opens a 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 organization. Explore and customize the pipeline as needed. Follow the steps below to familiarize yourself with the VSTS build and release pipelines.

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

pause a build, and edit the build pipeline.

3. Select **Edit**.
4. From this view, **examine the various tasks** for your build pipeline. The build performs various tasks such as fetching sources from the Git repository, restoring dependencies, and publishing outputs used for deployments.
5. At the top of the build pipeline, select the **build pipeline name**.
6. Change the **name** of your build pipeline to something more descriptive. Select **Save & queue**, then select **Save**.
7. Under your build pipeline name, select **History**. You see an audit trail of your recent changes for the build. VSTS keeps track of any changes made to the build pipeline, and allows you to compare versions.
8. Select **Triggers**. The Azure DevOps project automatically created a CI trigger, and every commit to the repository initiates a new build. You can optionally choose to include or exclude branches from the CI 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 pipeline to manage deployments to Azure.
11. On the left-hand side of the browser, select the **ellipsis** next to your release pipeline, then choose **Edit**.
12. The release pipeline contains a **pipeline**, which defines the release process. Under **Artifacts**, select **Drop**. The build pipeline you examined in the previous steps produces the output used for the artifact.
13. To the right-hand side of the **Drop** icon, select the **Continuous deployment trigger**. This release pipeline has an enabled CD trigger, which executes a deployment every time there is a new build artifact available. Optionally, you can disable the trigger, so your deployments require manual execution.
14. On the left-hand side of the browser, select **Tasks**. The tasks are the activities your deployment 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.

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 pipeline were automatically created in your VSTS project. You can modify these build and release pipelines to meet the needs of your team. To learn more see this tutorial:

Customize CD process

Videos

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

Build your ASP.NET 4 app

7/19/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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/adventworks/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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.

   - 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 pipeline.

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 pipeline by the template. These are the tasks that will automatically run every time you push code changes.

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 pipeline is automatically linked to your repository.

5. Click the **Triggers** tab in the build pipeline. 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 pipeline 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.

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 pipeline to meet the needs of your team. To learn more see one of these topics:

- CI builds for Git in VSTS

.NET Core

7/20/2018 • 11 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This guidance explains how to build .NET Core projects.

> **NOTE**
> This guidance applies to TFS version 2017.3 and newer.

> **NOTE**
> To use YAML you must have the **Build YAML definitions** preview feature enabled on your organization.

Example

If you want some sample code that works with this guidance, import (into VSTS or TFS) or fork (into GitHub) this repo:

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

- Designer
- YAML

> **IMPORTANT**
> If you are new to creating build pipelines, then complete the designer quickstart first before following these instructions.

- After you have the sample code in your own repository, create a build pipeline and select the **ASP.NET Core** template. This automatically adds the tasks required to build the code in the sample repository.
- Select **Process** under the **Tasks** tab in the build pipeline editor and change the properties as follows:
  - **Agent queue:** `Hosted Linux`
  - **Projects to test:** `**/*[Tt]ests/*.csproj`

Save the pipeline and queue a build to see it in action. Then read through the rest of this topic to learn some of the more common changes people make to customize a .NET Core build process.

Build environment

You can use VSTS to build your .NET Core projects on Windows, Linux, or macOS without needing to set up any infrastructure of your own. The Microsoft-hosted agents in VSTS have several released versions of the .NET Core SDKs preinstalled. Use the **Hosted VS2017** agent queue (to build on Windows), the **Hosted Linux Preview** agent queue, or the **Hosted macOS Preview** queue.

- Designer
- YAML

In the build pipeline, select **Tasks**, then select the **Process** node, and finally select the **Agent queue** that you want to use.

The Microsoft hosted agents do not include some of the older versions of the .NET Core SDK. They also do not typically include prerelease versions. If you need these kinds of SDKs and you prefer to use the Microsoft hosted agents, add the **.NET Core Tool Installer** task to the beginning of your process.

- Designer
- YAML

If you need a version of the .NET Core SDK that is not already installed on the Microsoft-hosted agent:

1. In the build pipeline, select **Tasks**, choose the phase that runs your build tasks, and then select **+** to add a new task to that phase.
2. In the task catalog, find and add the **.NET Core Tool Installer** task.
3. Select the task and specify the version of the .NET Core SDK or runtime that you want to install.

> **TIP**
> As an alternative to the approach shown above you can set up a self-hosted agent and save the cost of running the tool installer. You can also use self-hosted agents to save additional time if you have a large repository or you run incremental builds. For example, you don't select the "clean repository" option, or you don't use the `dotnet build --no-incremental` or `dotnet clean` commands.

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

Restore dependencies

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

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

You can download NuGet packages from NuGet.org.

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

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

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

If your builds occasionally fail when restoring packages from NuGet.org due to connection issues, you can use VSTS Package Management in conjunction with upstream sources, and cache the packages in VSTS. The credentials of the build pipeline are automatically used when connecting to VSTS Package Management. These credentials are derived from either the **Project Collection Build Service** account or the **Project Build Service** account depending on the option selected in your build pipeline.

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

If you're using Microsoft-hosted agents, you get a new machine every time your run a build - which means restoring the packages every time. This can take a significant amount of time. To mitigate this you can either use VSTS Package Management or a self-hosted agent, in which case you'll get the benefit of using the package cache.

To restore packages in your build pipeline:

- Designer
- YAML

1. Select **Tasks** in the build pipeline, select the phase that runs your build tasks, then select **+** to add a new task to that phase.
2. In the task catalog, find and add the **.NET Core** task.
3. Select the task and, for **Command**, select **restore**.
4. Specify any other options you need for this task, then save the build.

Build your project

You build your .NET Core project using the **.NET Core** task with the **Command** set to **build**. Running this task is equivalent to running `dotnet build` from a command prompt.

If you need to run additional .NET Core commands, you can specify other **Command** settings such as **clean** or **publish**. If you need a command not listed, you can set **Command** to **custom** and then type the name of the command in the **Custom command** box. This is useful, for example, to install dotnet global tools that are available in .NET Core 2.1.300 and newer.

- Designer
- YAML

Build

1. Select **Tasks** in the build pipeline, select the phase that runs your build tasks, then select **+** to add a new task to that phase.
2. In the task catalog, find and add the **.NET Core** task.
3. Select the task and, for **Command**, select **build** or **publish**.
4. Specify any other options you need for this task, then save the build.

Install a tool

To install a .NET Core global tool such as dotnetsay in your build running on Windows:

1. Add **.NET Core** task and set the following properties:

- **Command:** custom
- **Path to projects:** *leave empty*
- **Custom command:** tool
- **Arguments:** `install -g dotnetsay`

2. Add a **Command Line** task and set the following properties:

   - **Script:** `dotnetsay`

Run your tests

Use the **.NET Core** task to run unit tests in your .NET Core solution using testing frameworks such as MSTest, xUnit, and NUnit. One benefit of using this built-in task to run your tests is that the results of the tests are automatically published to VSTS or TFS. These results are then made available to you in the build summary.

- Designer
- YAML

Use the **.NET Core** task with **Command** set to **test**. **Path to projects** should refer to the test projects in your solution.

Package and deliver your code

The output of the `dotnet build` command is generally not transferable. To prepare your code to run on another machine, you must package the code.

- Designer
- YAML

Publish to a NuGet feed

If you want to publish your code to a NuGet feed:

1. Use a **.NET Core** task with the **Command** set to **pack**.

2. Publish your package to a NuGet feed.

Deploy a web app

If you want to deploy a web app:

1. Use a **.NET Core** task with the **Command** set to **publish**.

2. Make sure you've selected the option to create a .zip file archive.

When the build runs it will automatically search the folders in your solution for web projects, and produce .zip file packages for each of them.

Troubleshooting

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

- Check that the versions of the .NET Core SDK and runtime on your development machine match those on the agent. You can include a command line script `dotnet --version` in your build pipeline to print the version of .NET Core SDK. Either use the **.NET Core Tool Installer** (as explained in this guidance) to deploy the same version on the agent, or update your projects and development machine to the newer version of .NET Core SDK.

- You may be using some logic in Visual Studio IDE that is not encoded in your build pipeline. VSTS or TFS

run each of the commands you specify in the tasks one after the other in a new process. Look at the logs from the VSTS or TFS build to see the exact commands that ran as part of the build. Repeat the same commands in the same order on your development machine to locate the problem.

- If you have a mixed solution that includes some .NET Core projects and some .NET Framework projects, you should also use the **NuGet** task to restore packages specified in `package.json` files. Similarly, you should add **MSBuild** or **Visual Studio Build** tasks to build the .NET Framework projects.
- If your builds fail intermittently while restoring packages, either Nuget.org is having issues or there are networking problems between the Azure data center and Nuget.org. These are not under our control, and you may need to explore whether using VSTS Package Management with Nuget.org as an upstream source improves the reliability of your builds.
- Occasionally, when we roll out an update to the hosted images with a new version of .NET Core SDK or Visual Studio, something may break your build. This can happen, for example, if a newer version or feature of the NuGet tool is shipped with the SDK. To isolate these problems, use the **.NET Core Tool Installer** task to specify the version of the .NET Core SDK used in your build.

Q&A

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

Package Management in VSTS and TFS

Where can I learn more about .NET Core commands?

.NET Core command-line interface (CLI) tools

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

Unit testing in .NET Core projects

Where can I learn more about tasks?

Build and release tasks

Android

7/10/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This guide explains creating pipelines for Android projects. Before this guidance, read the designer quickstart or YAML quickstart.

Get started

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

- Designer
- YAML

Choose the **Android** template as you create a pipeline for your project. This will get you started with tasks to build, test, sign, and align an Android APK.

Sign and align an Android APK

Add the Android Signing task to sign and zipalign a built APK. An APK must be signed to run on a device instead of an emulator. Zipaligning reduces the RAM it consumes.

> **Important:** We recommend storing each of the specified passwords in a secret variable.

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

Test on the Android Emulator

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

Test on Azure-hosted devices

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

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

Retain artifacts

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

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

Deploy

App Center

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

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

Google Play

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

Release

Add the Google Play Release task to release a new Android app version to the Google Play store.

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

Promote

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

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

Increase rollout

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

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

Related extensions

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

Mobile Testing Lab (Perfecto Mobile)
React Native (Microsoft)

Build your Visual C++ app

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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/adventworks/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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.

   - 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 pipeline.

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 pipeline by the template. These are the tasks that will automatically run every time you push code changes.

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 pipeline 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 pipeline 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.

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.

Docker

7/20/2018 • 12 minutes to read • Edit Online

This guidance explains how to build Docker images.

> **NOTE**
> To use YAML you must have the **Build YAML definitions** preview feature enabled on your account.

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

> **NOTE**
> This guidance applies to TFS version 2017.3 and newer.

Example

To build a Docker image, you need a *Dockerfile*. If you want some sample code that includes this file and works with this guidance, then import (into VSTS or TFS) or fork (into GitHub) this repo:

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

- Designer
- YAML

> **IMPORTANT**
> If you are new to creating build pipelines, then complete the designer quickstart first before following these instructions.

> **NOTE**
> This scenario works on TFS, but some of the following instructions might not exactly match the version of TFS that you are using. Also, you'll need to set up a self-hosted agent, possibly also installing software. If you are a new user, you might have a better learning experience by trying this procedure out first using a free VSTS account. Then try this with VSTS.

1. After you have the sample code in your own repository, create a build pipeline and select the **ASP.NET Core** template. This automatically adds the tasks that you typically need to build an ASP.NET Core app.

2. Select **Process** under the **Tasks** tab of the build pipeline editor, and change its properties as follows:

- **Agent queue:** `Hosted Linux`
- **Projects to test:** `**/*[Tt]ests/*.csproj`

3. Modify the **.NET Core Publish** task in the build pipeline as follows:

   - **Arguments:** `--configuration $(BuildConfiguration) --output out`
   - **Zip published projects:** Clear this check box
   - **Add project name to publish path:** Clear this check box

4. Remove the **Publish artifact** task.

5. Add **Docker** task after the **.NET Core Publish** task and configure it as follows to build an image using the **Dockerfile** in the repository:

   - **Action:** `Build an image`

6. Add another **Docker** task and configure it as follows to push the image to your Docker hub registry:

   - **Action:** `Push an image`
   - **Container registry type:** `Container registry`
   - **Docker registry connection:** Select `New` and create a connection to your Docker hub registry.

Save the pipeline and queue a build to see it in action.

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

Build environment

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

- Designer
- YAML

In the build pipeline, select **Tasks**, then select the **Process** node, and finally select the **Agent queue** that you want to use.

Microsoft-hosted Linux agents

Use the **Hosted Linux** agent queue to build Linux container images. When you use this queue, you get a fresh Linux virtual machine with each build. This virtual machine runs the agent and acts as a Docker host. Tasks in your build do not directly run on the virtual machine at present. Instead, they run in a Microsoft-provided Docker container on the virtual machine. Shared volumes are used to faciliate communication between the virtual machine and the container. You can run Docker commands as part of your build, since the `docker.sock` socket of the host is volume mounted in the container.

You cannot use **Hosted Mac** to build container images as Docker is not installed on these agents.

Microsoft-hosted VS2017 (Windows) agents

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

> **NOTE**
> - Using Hosted VS2017 agents, you can only build Docker images with Windows Server 2016 as the container OS. You cannot build Docker images with Windows Server 1803 as the container OS since the host operating system on the virtual machines is Windows Server 2016.
> - We do not yet have a pool of Microsoft-hosted agents running Windows Server 1803. Until this is available, you can build Windows Server 1803 images using self-hosted agents.

Self-hosted agents

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

Your builds run on a self-hosted agent. Make sure that you have Docker installed on the agent.

Build an image

You can build a Docker image by running the `docker build` command in a script or by using the **Docker** task.

- Designer
- YAML

1. Select **Tasks** in your build pipeline, and then add the **Docker** task to the phase.
2. Select the task, and then for **Action**, select **Build an image**.
3. If your *Dockerfile* depends on an image that is present in an authenticated registry (for example, a private Docker registry or Azure Container Registry), then specify those properties in the **Container registry type** and the corresponding service connection.

Integrate build and test tasks

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

Build and test in your build pipeline

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

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

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

The instructions in the above example demonstrate this approach.

Build and test in your Dockerfile

In this approach, you use your *Dockerfile* to build your code and run tests. The build pipeline has a single step to run `docker build`. The rest of the steps are orchestrated by the Docker build process. It's common to use a multi-stage Docker build in this approach. The advantage of this approach is that your build process is entirely configured in your *Dockerfile*. This means your build process is portable from the development machine to any build system. One disadvantage is that you can't leverage VSTS and TFS features such as tasks, phases, or test

analytics.

To follow this approach, create a *Dockerfile* at the root of your repo with the following content:

```
# First stage of multi-stage build
FROM microsoft/aspnetcore-build:2.0 AS build-env
WORKDIR /app

# copy the contents of agent working directory on host to workdir in container
COPY . ./

# dotnet commands to build, test, and publish
RUN dotnet restore
RUN dotnet build -c Release
RUN dotnet test dotnetcore-tests/dotnetcore-tests.csproj -c Release
RUN dotnet publish -c Release -o out

# Second stage - Build runtime image
FROM microsoft/aspnetcore:2.0
WORKDIR /app
COPY --from=build-env /app/dotnetcore-sample/out .
ENTRYPOINT ["dotnet", "dotnetcore-sample.dll"]
```

Then, define your build pipeline:

- Designer
- YAML

1. Select **Tasks** in the build pipeline, and then remove all the tasks.
2. Add a **Docker** task, and then for **Action** select **Build an image**.

Push an image

Once you've built a Docker image, you can use the **Docker** task to push it to a Docker registry or to Azure Container Registry (ACR). The **Docker** task makes the process easier for you because it sets up an authenticated connection to your registry or ACR.

To push a Docker image:

- Designer
- YAML

1. In your build pipeline, select **Tasks**, and then add a **Docker** task to the phase that runs your build tasks.
2. Select the **Docker** task, and then for **Action** select **Push an image**.
3. Specify how to connect to your registry in the **Container registry type** and the corresponding service connection properties.

By default, the **Docker** task tags your image as `<Docker id>/<repo name>:<build id>`

Use Docker Compose

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

If you use Microsoft-hosted agents, you do not have to run any additional steps to install and use docker-compose.

To extend the above example to use docker-compose:

1. Add `docker-compose.yml` file at the root of your repo.

```
sut:
build: .
dockerfile: Dockerfile.test
links:
 - web
web:
build: .
dockerfile: Dockerfile
```

2. Add a `Dockerfile.test` file at the root of your repo.

```
FROM ubuntu:trusty
RUN apt-get update && apt-get install -yq curl && apt-get clean
WORKDIR /app
ADD test.sh /app/test.sh
CMD ["bash", "test.sh"]
```

3. Add a `test.sh` file at the root of your repository.

```
sleep 5
if curl web | grep -q 'ASP.NET Core '; then
echo "Tests passed!"
exit 0
else
echo "Tests failed!"
exit 1
fi
```

4. Add a **Bash** task.

- Designer
- YAML

In the build pipeline, add a **Bash** task with the following inline script:

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

> **NOTE**
> When using Hosted Linux agents, the agent runs inside a container. The network of this container is not bridged to the network of the containers that you spin up through docker compose. As a result, you cannot communicate from the agent to one of the containers in the composition, for example, to drive tests. One way to solve this problem is to explicitly create another test driver as a container within the composition, as we did in the example above. Another solution is to use `docker-compose exec` and target a specific container in the composition from your script.

Build ARM containers

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

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

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

2. Run the following script in your build pipeline.

```
# register QEMU binary - this can be done by running the following Docker image
docker run --rm --privileged multiarch/qemu-user-static:register --reset
# build your image
docker build -t <your-image-tag> .
```

Troubleshooting

If you're able to build your image on your development machine, but are having trouble building it on VSTS or TFS, the following solutions might help:

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

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

Build your Go app

7/19/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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/adventworks/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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.

- 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 pipeline.

VSTS

1. In the right panel, search for `Go`, select **Go (Preview)**, and then click **Apply**.

   You now see all the tasks that were automatically added to the build pipeline by the template. These are the tasks that will automatically run every time you push code changes.

2. For the **Agent queue**:

   - **VSTS:** Select *Hosted Linux*, *Hosted macOS*, or *Hosted VS2017*. This will use a Microsoft-hosted agent with Go installed.
   - **TFS:** Select a queue that includes an agent with Go installed.

3. Click **Get sources** and observe that the new build pipeline is automatically linked to your repository.

TFS 2018 and 2017.2

1. In the right panel, select **Empty**, and then click **Apply**. This template allows starting from scratch and adding your own build tasks.
2. For the **Agent queue**, select a queue that includes an agent with Go installed.
3. Click **Get sources** and observe that the new build pipeline is automatically linked to your repository.
4. Click the **+** icon on **Phase 1** of the build.
5. Search for the **Command Line** task and click **Add** to add it to your build.
6. Click the **Command Line** task and set its field values as follows:

FIELD	VALUE
Display name	`Build Go application`
Tool	`go`
Arguments	`build hello.go`

Finish

1. Click the **Triggers** tab in the build pipeline. Enable the **Continuous integration** trigger. This will ensure that the build process is automatically triggered every time changes are pushed to your repository.
2. Click **Save & queue** to kick off your first build. On the **Save build pipeline and queue** dialog box, click **Save & queue**.
3. 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 pipeline 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

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization.

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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.
   - 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 pipeline.

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 pipeline by the template. These are the steps that will automatically run every time you push code changes.

3. For the **Agent queue**:

   - **VSTS:** Select *Hosted Linux*, *Hosted macOS*, or *Hosted VS2017*. This will use a Microsoft-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 pipeline is automatically linked to your repository.

5. Click the **Triggers** tab in the build pipeline. Enable the **Continuous integration** trigger. This will ensure that the build process is automatically triggered every time changes are pushed to your repository.

6. Click **Save & queue** to kick off your first build. On the **Save build pipeline 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.

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 pipeline 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

7/20/2018 • 10 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.3

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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 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 use 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 pipeline

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 pipeline that helps your team keep the master branch clean.

Begin by creating your build pipeline.

- 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 pipeline.

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

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 pipeline by the template. These are the tasks that will automatically run every time you push code changes.

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 pipeline.

Finish the CI process pipeline

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*. 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 pipeline is automatically linked to your repository.

3. Select the **Triggers** tab in the build pipeline. 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 pipeline 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

Python

7/13/2018 • 3 minutes to read • Edit Online

This guide explains creating pipelines for Python projects. Before this guidance, read the YAML quickstart or designer quickstart.

Get started

You can build Python projects using Microsoft-hosted agents that include tools for Python. Or, you can use self-hosted agents with specific tools you need.

- YAML
- Designer

Create a file named **.vsts-ci.yml** in the root of your repository. Then, add applicable phases and tasks to the YAML file as described below.

Use a specific Python version

Add the Use Python Version task to set the version of Python used in your pipeline. This sample sets subsequent pipeline tasks to use Python 3.6.

```
# https://aka.ms/yaml
queue: 'Hosted Linux'
steps:

- task: UsePythonVersion@0
  inputs:
    versionSpec: '3.6'
    architecture: 'x64'
```

Use multiple Python versions

To run a pipeline with multiple Python versions, such as to test your project using different versions, define a phase with a matrix of Python version values. Then set the Use Python Version task to reference the matrix variable for its Python version. Increase the **parallel** value to simultaneously run the phase for all versions in the matrix, depending on how many concurrent jobs are available.

```
# https://aka.ms/yaml
phases:
- phase: 'Test'
  queue:
    name: 'Hosted Linux Preview'
    parallel: 1
    matrix:
      Python27:
        python.version: '2.7'
      Python35:
        python.version: '3.5'
      Python36:
        python.version: '3.6'
  steps:

  - task: UsePythonVersion@0
    inputs:
      versionSpec: '$(python.version)'
      architecture: 'x64'

  # Add additional tasks to run using each Python version in the matrix above
```

Activate an Anaconda environment

As an alternative to the **Use Python Version** task, create and activate a conda environment and Python version using the Conda Environment task. Add the following YAML to activate an environment named `myEnvironment` with the Python 3 package.

```
- task: CondaEnvironment@0
  inputs:
    environmentName: 'myEnvironment'
    packageSpecs: 'python=3'
```

Run a Python script

Run a Python script in your pipeline by adding the Python Script task. The script can be defined in a file or in-line with the task.

Add the following YAML to run a Python script file named `myPythonScript.py`.

```
- task: PythonScript@0
  inputs:
    targetType: 'filePath'
    filePath: 'src/myPythonScript.py'
    arguments: ''
```

Alternatively, set the **targetType** to `inline` to define the script in YAML.

```
- task: PythonScript@0
  inputs:
    targetType: 'inline'
    script: |
      print('Hello world 1')
      print('Hello world 2')
    arguments: ''
```

Install dependencies

Install requirements with pip

Add the following YAML to install or upgrade `pip` and requirements specified in `requirements.txt`.

```
- script: python -m pip install --upgrade pip && pip install -r requirements.txt
  displayName: 'Install requirements'
```

Install specific PyPI packages with pip

Add the following YAML to install or upgrade `pip` and two specific packages: `setuptools` and `wheel`.

```
- script: python -m pip install --upgrade pip setuptools wheel
  displayName: 'Install dependencies'
```

Install Anaconda packages with conda

Add the following YAML to install the `scipy` package in the conda environment named `myEnvironment`. See Activate an Anaconda environment above.

```
- script: conda install -n myEnvironment scipy
  displayName: 'Install conda libraries'
```

Test

Test with pytest

Add the following YAML to install `pytest`, run tests, and output results in JUnit format.

```
- script: pip install pytest && pytest tests --doctest-modules --junitxml=junit/test-results.xml
  displayName: 'pytest'
```

Publish test results

Add the Publish Test Results task to publish JUnit or xUnit test results to the server.

```
- task: PublishTestResults@2
  inputs:
    testResultsFiles: '**/test-*.xml'
    testRunTitle: 'Test results for Python $(python.version)'
```

Deploy

Deploy to a PyPI-compatible index

Add the PyPI Publisher task to package and publish to a PyPI-compatible index.

```
- task: PyPIPublisher@0
  inputs:
    pypiConnection: ''
    packageDirectory: '$(build.sourcesDirectory)'
    alsoPublishWheel: false
```

Retain artifacts

Add the Publish Build Artifacts task to store your build output with the build record or test and deploy it in subsequent pipelines. See Artifacts.

```
- task: PublishBuildArtifacts@1
  pathToPublish: 'dist'
```

Related extensions

Python Build Tools (for Windows) (Steve Dower)
PyLint Checker (Darren Fuller)
Python Test (Darren Fuller)
VSTS Plugin for PyCharm (IntelliJ) (Microsoft)
Python extension for Visual Studio Code (Microsoft)
VSTS extension for Visual Studio Code (Microsoft)

Build your Xamarin app

7/19/2018 • 6 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization.
- You will build the sample app for Android and iOS using two build pipelines in this quickstart. If you use VSTS, you can use a Microsoft-hosted agent for both. If you use TFS, you will need a self-hosted agent to build Xamarin.Android and Xamarin.iOS. Xamarin.iOS requires an agent running on macOS. Set up a self-hosted 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	MICROSOFT-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 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 pipeline that helps your team keep the master branch clean.

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

Define your Xamarin.Android build

1. Create a new build pipeline.

   - 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 pipeline.

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 pipeline by the template. These are the tasks that will automatically run every time you push code changes.

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 pipeline 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 pipeline by right-clicking it and selecting **Remove selected task(s)**.
8. Click **Save & queue** to kick off your first build. On the **Save build pipeline 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 pipeline. This time, select the **Xamarin.iOS** template.

1. For the **Agent queue**, select a hosted macOS queue such as **Hosted macOS**, 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 pipeline 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 task 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

7/19/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, 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** 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 Node.js, 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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.

   - 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 pipeline.

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

   You now see all the tasks that were automatically added to the build pipeline by the template. These are the tasks that will automatically run every time you push code changes.

3. For the **Agent queue**, select **Hosted macOS** 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 tasks 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 pipeline is automatically linked to your repository.

7. Click the **Triggers** tab in the build pipeline. 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 pipeline 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 accidently kill other processes.

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 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 xcsharedata folder to source control).

2 minutes to read

Deploy apps to Azure Stack

7/10/2018 • 2 minutes to read • Edit Online

Azure Stack is an extension of Azure that enables the agility and fast-paced innovation of cloud computing through a hybrid cloud and on-premises environment.

> At present, Team Foundation Server can be used to deploy to Azure Stack with Azure AD and cannot be used to deploy to an Azure Stack with Azure Directory Federated Services (AD FS). Azure Stack with AD FS requires a service principal with certificate, which is not currently supported in VSTS/TFS.

To enable connection to an Azure Stack, you specify it as the **Environment** parameter when you create an Azure Resource Manager service connection. You must use the full version of the service connection dialog to manually define the connection. Before you configure a service connection, you should also ensure you meet all relevant compliance requirements for your application.

You can then use the service connection in your build and release pipeline tasks.

Next

- Deploy an Azure web app
- Troubleshoot Azure Resource Manager service connections
- Azure Stack Operator Documentation

Q&A

Are all the Azure tasks supported?

The following Azure tasks are validated with Azure Stack:

- Azure PowerShell
- Azure File Copy
- Azure Resource Group Deployment
- Azure App Service Deploy
- Azure App Service Manage
- Azure SQL Database Deployment

How do I resolve SSL errors during deployment?

To ignore SSL errors, set a variable named `VSTS_ARM_REST_IGNORE_SSL_ERRORS` to the value `true` in the build or release pipeline.

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Azure SQL database deployment

7/17/2018 • 7 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can automatically deploy your database updates to Azure SQL database after every successful build. Before you read this topic, you should understand the type of pipeline that you're creating: designer or YAML.

> **NOTE**
> To use YAML you must have the **Build YAML pipelines** preview feature enabled.

DACPAC

The simplest way to deploy a database is to create data-tier package or DACPAC. DACPACs can be used to package and deploy schema changes as well as data. You can create a DACPAC using the **SQL database project** in Visual Studio.

- Designer
- YAML

When setting up a build pipeline for your Visual Studio database project, use the **.NET desktop** template. This template automatically adds the tasks to build the project and publish artifacts, including the DACPAC.

When setting up a release pipeline, choose **Start with an empty process**, link the artifacts from build, and then add an Azure SQL Database Deployment task.

SQL scripts

Instead of using a DACPAC, you can also use SQL scripts to deploy your database. Here is a simple example of a SQL script that creates an empty database.

```
USE [master]
GO
IF NOT EXISTS (SELECT name FROM master.sys.databases WHERE name = N'DatabaseExample')
CREATE DATABASE [DatabaseExample]
GO
```

To run SQL scripts as part of a VSTS pipeline, you will need Azure Powershell scripts to create and remove firewall rules in Azure. Without the firewall rules, the VSTS agent cannot communicate with Azure SQL Database.

The following Powershell script creates firewall rules. You can check-in this script as `SetAzureFirewallRule.ps1` into your repository.

```
[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
  [String] [Parameter(Mandatory = $true)] $ServerName,
  [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

function New-AzureSQLServerFirewallRule {
  $agentIP = (New-Object net.webclient).downloadstring("http://checkip.dyndns.com") -replace "[^\d\.]"
  New-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIp -EndIPAddress $agentIp -RuleName
$AzureFirewallName -ServerName $ServerName
}
function Update-AzureSQLServerFirewallRule{
  $agentIP= (New-Object net.webclient).downloadstring("http://checkip.dyndns.com") -replace "[^\d\.]"
  Set-AzureSqlDatabaseServerFirewallRule -StartIPAddress $agentIp -EndIPAddress $agentIp -RuleName
$AzureFirewallName -ServerName $ServerName
}

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -RuleName $AzureFirewallName -ErrorAction
SilentlyContinue) -eq $null)
{
  New-AzureSQLServerFirewallRule
}
else
{
  Update-AzureSQLServerFirewallRule
}
```

The following Powershell script removes firewall rules. You can check-in this script as `RemoveAzureFirewall.ps1` into your repository.

```
[CmdletBinding(DefaultParameterSetName = 'None')]
param
(
  [String] [Parameter(Mandatory = $true)] $ServerName,
  [String] $AzureFirewallName = "AzureWebAppFirewall"
)

$ErrorActionPreference = 'Stop'

If ((Get-AzureSqlDatabaseServerFirewallRule -ServerName $ServerName -RuleName $AzureFirewallName -ErrorAction
SilentlyContinue))
{
  Remove-AzureSqlDatabaseServerFirewallRule -RuleName $AzureFirewallName -ServerName $ServerName
}
```

- Designer
- YAML

When you set up a build pipeline, make sure that the SQL script to deploy the database and the Azure powershell scripts to configure firewall rules are part of the build artifact.

When you set up a release pipeline, choose **Start with an Empty process**, link the artifacts from build, and then use the following tasks:

- First, use an Azure Powershell task to add a firewall rule in Azure to allow the VSTS agent to connect to Azure SQL Database. The script requires one argument - the name of the SQL server you created.
- Second, use a Command line task to run the SQL script using the **SQLCMD** tool. The arguments to this tool are
  `-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`.

- Third, use another Azure Powershell task to remove the firewall rule in Azure.

Azure service connection

The **Azure SQL Database Deployment** task is the primary mechanism to deploy a database to Azure. This task, as with other built-in Azure tasks, requires an Azure service connection as an input. The Azure service connection stores the credentials to connect from VSTS or TFS to Azure.

The easiest way to get started with this task is to be signed in as a user that owns both the VSTS and the Azure subscriptions. In this case, you won't have to manually create the service connection. Otherwise, to learn how to create an Azure service connection, see Create an Azure service connection.

To learn how to create an Azure service connection, see Create an Azure service connection.

Deploying conditionally

You may choose to deploy only certain builds to your Azure database.

- Designer
- YAML

In your release pipeline you can implement various checks and conditions to control the deployment.

- Set **branch filters** to configure the **continuous deployment trigger** on the artifact of the release pipeline.
- Set **pre-deployment approvals** as a pre-condition for deployment to an environment.
- Configure **gates** as a pre-condition for deployment to an environment.
- Specify conditions for a task to run.

To learn more, see Release, branch, and environment triggers, Release deployment control using approvals, Release deployment control using gates, and Specify conditions for running a task.

Additional SQL actions

SQL Azure Dacpac Deployment may not support all SQL server actions that you want to perform. In these cases, you can simply use Powershell or command line scripts to run the commands you need. This section shows some of the common use cases for invoking the `SqlPackage.exe` tool. As a prerequisite to running this tool, you must use a self-hosted agent and have the tool installed on your agent.

> **NOTE**
> If you execute **SQLPackage** from the folder where it is installed, you must prefix the path with `&` and wrap it in double-quotes.

Basic Syntax

```
<Path of SQLPackage.exe> <Arguments to SQLPackage.exe>
```

You can use any of the following SQL scripts depending on the action that you want to perform

Extract

Creates a database snapshot (.dacpac) file from a live SQL server or Microsoft Azure SQL Database.

Command Syntax:

```
SqlPackage.exe /TargetFile:"<Target location of dacpac file>" /Action:Extract
/SourceServerName:"<ServerName>.database.windows.net"
/SourceDatabaseName:"<DatabaseName>" /SourceUser:"<Username>" /SourcePassword:"<Password>"
```

or

```
SqlPackage.exe /action:Extract /tf:"<Target location of dacpac file>"
/SourceConnectionString:"Data Source=ServerName;Initial Catalog=DatabaseName;Integrated Security=SSPI;Persist
Security Info=False;"
```

Example:

```
SqlPackage.exe /TargetFile:"C:\temp\test.dacpac" /Action:Extract
/SourceServerName:"DemoSqlServer.database.windows.net"
 /SourceDatabaseName:"Testdb" /SourceUser:"ajay" /SourcePassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Extract /?
```

Publish

Incrementally updates a database schema to match the schema of a source .dacpac file. If the database does not exist on the server, the publish operation will create it. Otherwise, an existing database will be updated.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Dacpac file location>" /Action:Publish /TargetServerName:"
<ServerName>.database.windows.net"
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password> "
```

Example:

```
SqlPackage.exe /SourceFile:"E:\dacpac\ajyadb.dacpac" /Action:Publish
/TargetServerName:"DemoSqlServer.database.windows.net"
/TargetDatabaseName:"Testdb4" /TargetUser:"ajay" /TargetPassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Publish /?
```

Export

Exports a live database, including database schema and user data, from SQL Server or Microsoft Azure SQL Database to a BACPAC package (.bacpac file).

Command Syntax:

```
SqlPackage.exe /TargetFile:"<Target location for bacpac file>" /Action:Export /SourceServerName:"
<ServerName>.database.windows.net"
/SourceDatabaseName:"<DatabseName>" /SourceUser:"<Username>" /SourcePassword:"<Password>"
```

Example:

```
SqlPackage.exe /TargetFile:"C:\temp\test.bacpac" /Action:Export
/SourceServerName:"DemoSqlServer.database.windows.net"
/SourceDatabaseName:"Testdb" /SourceUser:"ajay" /SourcePassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Export /?
```

Import

Imports the schema and table data from a BACPAC package into a new user database in an instance of SQL Server or Microsoft Azure SQL Database.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Bacpac file location>" /Action:Import /TargetServerName:"
<ServerName>.database.windows.net"
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password>"
```

Example:

```
SqlPackage.exe /SourceFile:"C:\temp\test.bacpac" /Action:Import
/TargetServerName:"DemoSqlServer.database.windows.net"
/TargetDatabaseName:"Testdb" /TargetUser:"ajay" /TargetPassword:"SQLPassword"
```

Help:

```
sqlpackage.exe /Action:Import /?
```

DeployReport

Creates an XML report of the changes that would be made by a publish action.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Dacpac file location>" /Action:DeployReport /TargetServerName:"
<ServerName>.database.windows.net"
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password>" /OutputPath:"<Output
XML file path for deploy report>"
```

Example:

```
SqlPackage.exe /SourceFile:"E: \dacpac\ajyadb.dacpac" /Action:DeployReport
/TargetServerName:"DemoSqlServer.database.windows.net"
/TargetDatabaseName:"Testdb" /TargetUser:"ajay" /TargetPassword:"SQLPassword"
/OutputPath:"C:\temp\deployReport.xml"
```

Help:

```
sqlpackage.exe /Action:DeployReport /?
```

DriftReport

Creates an XML report of the changes that have been made to a registered database since it was last registered.

Command Syntax:

```
SqlPackage.exe /Action:DriftReport /TargetServerName:"<ServerName>.database.windows.net" /TargetDatabaseName:"
<DatabaseName>"
/TargetUser:"<Username>" /TargetPassword:"<Password>" /OutputPath:"<Output XML file path for drift report>"
```

Example:

```
SqlPackage.exe /Action:DriftReport /TargetServerName:"DemoSqlServer.database.windows.net"
/TargetDatabaseName:"Testdb"
/TargetUser:"ajay" /TargetPassword:"SQLPassword" /OutputPath:"C:\temp\driftReport.xml"
```

Help:

```
sqlpackage.exe /Action:DriftReport /?
```

Script

Creates a Transact-SQL incremental update script that updates the schema of a target to match the schema of a source.

Command Syntax:

```
SqlPackage.exe /SourceFile:"<Dacpac file location>" /Action:Script /TargetServerName:"
<ServerName>.database.windows.net"
/TargetDatabaseName:"<DatabaseName>" /TargetUser:"<Username>" /TargetPassword:"<Password>" /OutputPath:"<Output
SQL script file path>"
```

Example:

```
SqlPackage.exe /Action:Script /SourceFile:"E:\dacpac\ajyadb.dacpac"
/TargetServerName:"DemoSqlServer.database.windows.net"
/TargetDatabaseName:"Testdb" /TargetUser:"ajay" /TargetPassword:"SQLPassword" /OutputPath:"C:\temp\test.sql"
/Variables:StagingDatabase="Staging DB Variable value"
```

Help:

```
sqlpackage.exe /Action:Script /?
```

Azure web app deployment

7/17/2018 • 9 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can automatically deploy your web app to an Azure App Services web app after every successful build. Before you read this topic, you should understand the type of pipeline that you're creating: designer or YAML.

> **NOTE**
> This guidance applies to TFS version 2017.3 and later.

> **NOTE**
> To use YAML you must have the **Build YAML pipelines** preview feature enabled.

Example

If you want some sample code that works with this guidance, import (into VSTS or TFS) or fork (into GitHub) this repo:

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

Follow the guidance in .NET Core to build the sample code.

- Designer
- YAML

After you have a build, create a release pipeline and select the **Azure App Service Deployment** template for your environment. This automatically adds the necessary tasks. Link the build as an artifact to this release pipeline. Save the pipeline and create a release to see it in action. Then read through the rest of this topic to learn some of the more common changes that people make to customize an Azure web app deployment.

Azure App Service Deploy task

- Designer
- YAML

The simplest way to deploy to an Azure web app is to use the **Azure App Service Deploy** task. This task is automatically added to the release pipeline when you select one of the pre-built deployment templates for Azure app service deployment. Templates exist for apps developed in various programming languages. If you cannot find a template for your language, select the generic **Azure App Service Deployment** template.

When you link the artifact in your release pipeline to a build that compiles and publishes the web package, it is automatically downloaded and placed into the `$(System.ArtifactsDirectory)` folder on the agent as part of the

release. This is where the task picks up the web package for deployment.

Azure service connection

The **Azure App Service Deploy** task, similar to other built-in Azure tasks, requires an Azure service connection as an input. The Azure service connection stores the credentials to connect from VSTS or TFS to Azure.

- Designer
- YAML

The easiest way to get started with this task is to be signed in as a user that owns both the VSTS and the Azure subscriptions. In this case, you won't have to manually create the service connection. Otherwise, to learn how to create an Azure service connection, see Create an Azure service connection.

To learn how to create an Azure service connection, see Create an Azure service connection.

Deploy to a virtual application

- Designer
- YAML

By default, your deployment happens to the root application in the Azure web app. If you want to deploy to a specific virtual application, type its name in the **Virtual Application** property of the **Azure App Service Deploy** task.

Deploy to a slot

- Designer
- YAML

You can configure the Azure web app to have multiple slots. Slots allow you to safely deploy your app and test it before making it available to your customers. Use the option **Deploy to Slot** in the **Azure App Service Deploy** task to specify the slot to deploy to. You can swap the slots by using the **Azure App Service Manage** task.

Deploy to multiple web apps

- Designer
- YAML

If you want to deploy to multiple web apps, add environments to your release pipeline. You can control the order of deployment. To learn more, see Environments.

Configuration changes

You may want to apply a specific configuration for your web app target before deploying to it. This is particularly useful when you deploy the same build to multiple web apps in a pipeline. For example, if your **Web.config** file contains a connection string named `connectionString`, you can change its value before deploying to each web app. You can do this either by applying a web config transformation or by substituting variables in your Web.config file.

- Designer
- YAML

To change the `connectionString` using variable substitution:

1. Create a release pipeline with two environments.
2. Link the artifact of the release to the build that produces the web package.

3. Define `connectionString` as a variable in each of the environments. Set the appropriate value.
4. Select the **XML variable substitution** option under the **File Transforms and Variable Substitution Options** of the **Azure App Service Deploy** task.

Deploying conditionally

You may choose to deploy only certain builds to your Azure web app.

- Designer
- YAML

In your release pipeline you can implement various checks and conditions to control the deployment.

- Set **branch filters** to configure the **continuous deployment trigger** on the artifact of the release pipeline.
- Set **pre-deployment approvals** as a pre-condition for deployment to an environment.
- Configure **gates** as a pre-condition for deployment to an environment.
- Specify conditions for a task to run.

To learn more, see Release, branch, and environment triggers, Release deployment control using approvals, Release deployment control using gates, and Specify conditions for running a task.

Deploy to a Government cloud or Azure Stack

Do this by creating a suitable service connection:

- Azure Government Cloud deployment
- Azure Stack deployment

Deployment mechanisms

The examples above rely on the built-in Azure App Service Deploy task, which provides simplified integration with Azure.

If you use a Windows agent, this task uses Web Deploy technology to interact with the Azure web app. Web Deploy provides several convenient deployment options such as renaming locked files, excluding files from the App_Data folder during deployment, and so on.

If you use the Linux agent, the task relies on the Kudu REST APIs.

The Azure App Service Manage task is another task that's useful for deployment. You can use this task to start, stop, or restart the web app prior to or after deployment. This task can also be used to swap slots, install site extensions, or enable monitoring of the web app.

If the built-in tasks do not meet your needs, you can use other methods to script your deployment. View the YAML snippets in each of the following tasks for some examples:

- Azure Powershell task
- Azure CLI task
- FTP task

Deploy to a Linux Virtual Machine

7/9/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

We'll show you how to set up continuous deployment of your app to an nginx web server running on Ubuntu using 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, 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**.

   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.

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 pipeline:

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

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

2. Choose **Start with an Empty process**.

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

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter that includes the **master** branch.

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

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

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

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

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.

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
```

10. Save the release pipeline.

Create a release to deploy your app

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

1. Choose **+ Release** and select **Create Release**.
2. Check that the build you just completed is selected 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 app and verify its contents.

Next steps

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

Publish npm packages

7/9/2018 • 2 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can publish npm packages produced by your build to:

- The VSTS or TFS Package Management service
- Other registries such as `https://registry.npmjs.org/`

Before you read this topic, you should understand the kind of build pipeline you're creating: designer or YAML.

- Web
- YAML

To publish an npm package to a Package Management feed, add the **npm** task. Then, 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`

To publish to a VSTS Package Management feed, set the identity under which the build runs as a **Contributor** on the feed. If your build pipeline is configured to run using the project collection scope, then give the **Project Collection Build Service** identity contributor access to the feed. If your build pipeline is configured to run using project scope, then the **Project Build Service** identity should be given contributor access to the feed. You can check the scope in which your builds run by opening the editor for your build pipeline, and looking at the **Options** tab. To learn more about permissions for Package Management feeds, see Secure and share packages using feed permissions .

To publish to a external npm registry, you must first create a service connection to point to that registry. You can do this by going to **Project settings**, then choosing **Service connections**, and then creating a **New service connection**. Select the **npm** option for the service connection. Fill in registry URL and the credentials to connect to the registry.

> **NOTE**
> Build does not support using the `publishConfig` property to specify the `registry` to which you're publishing. Ensure your working folder has a `.npmrc` file with a `registry=` line, as detailed in the Connect to feed screen in your feed.

Q&A

Where can I learn about the VSTS and TFS Package management service?

Package Management service

Publish to NuGet feeds

7/19/2018 • 7 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can publish NuGet packages from your build to NuGet feeds. You can publish these packages to

- The VSTS or TFS Package Management service
- Other NuGet services such as NuGet.org
- Your internal NuGet repository

Before you read this topic, you should understand the kind of build pipeline you're creating: designer or YAML.

Create a NuGet package

There are a variety of ways to create NuGet packages during a build. If you're already using MSBuild or some other task to create your packages, skip this section to go ahead and publish your packages. Otherwise, add a **NuGet** task:

- Web
- YAML

Add the **NuGet** task to your build in order to create a NuGet package. Make sure to add this task below the task that builds your application and above any tasks that require the packages you build.

The NuGet task supports a number of options. Some of the key ones are described below. The rest are described in the task documentation.

- **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 task.
- **Pack options > Use Build number to version package:** See the next section

Package versioning

In NuGet, a particular package is identified by its name and version number. A recommended approach to versioning packages is to use semantic versioning. Semantic version numbers have 3 numeric components, `Major.Minor.Patch`. When you fix a bug, you increment the patch (`1.0.0` → `1.0.1`). When you release a new backwards-compatible feature, you increment minor and reset patch to 0 (`1.4.17` → `1.5.0`). When you make a backwards-incompatible change, you increment major and reset minor and patch to 0 (`2.6.5` → `3.0.0`).

In addition to `Major.Minor.Patch`, semantic versioning provides for a prerelease label. Prerelease labels are a "-" followed by whatever letters and numbers you want. Version `1.0.0-alpha`, `1.0.0-beta`, and `1.0.0-foo12345` are all prerelease versions of `1.0.0`. Even better, semantic versioning specifies that when you sort by version number, those prerelease versions fit exactly where you'd expect: `0.99.999` < `1.0.0-alpha` < `1.0.0` < `1.0.1-beta`.

When you create a package in CI, you can use semantic versioning with prerelease labels. The **NuGet** task can be used for this purpose, and supports the following formats:

- Use the same versioning scheme for your builds and packages, provided that scheme has at least three parts separated by periods. The following build pipeline formats are examples of versioning schemes that are compatible with NuGet.
  - `$(Major).$(Minor).$(rev:.r)`, where `Major` and `Minor` are two variables defined in the build pipeline. This will automatically increment the build number and the package version with a new patch number keeping the major and minor versions constant, until you change them manually in the build pipeline.
  - `$(Major).$(Minor).$(Patch).$(date:yyyyMMdd)`, where `Major`, `Minor`, and `Patch` are variables defined in the build pipeline. This will create a new prerelease label for the build and package while keeping the major, minor, and patch versions constant.
- Use a version that is different from the build number. You can customize the major, minor, and patch versions for your packages in the NuGet task, and let the task generate a unique prerelease label based on date and time.

- Use a script in your build process to generate the version.

- Web
- YAML

In the **NuGet** task, choose **Pack options** and select one of the values for **Automatic package versioning**.

While semantic versioning with prerelease labels is a good solution for packages produced in CI builds, including a prerelease label is not ideal when you want to release a package to your users. The challenge is that packages once produced are immutable and so cannot be updated or replaced. When you're producing a package in build, you can't know whether it will be the version that you aim to release to your users or just a step along the way towards that release. While none of the following solutions are ideal, you can use one of these depending on your preference:

- Once you validate a package and decide to release it, produce another package without the prerelease label and publish it. The drawback of this approach is that you have to validate the new package again, and it may uncover new issues.

- Publish only packages that you wish to release. In this case, you will not use a prerelease label for every build. Instead, you will reuse the same package version for all packages. Since you do not publish packages from every build, you do not cause a conflict.

Publish your packages

In the previous section, you learned how to create a package with every build. When you are ready to share the changes to your package with your users, you can publish it.

- Web
- YAML

To publish NuGet packages created by your build, add the **NuGet** task and configure these options:

- **Command:** push
- **Path to NuGet package(s) to publish:** Leave this set to `$(Build.ArtifactStagingDirectory)` unless you decided earlier to pack your packages in another location in the last task.
- **Target feed location:** You can publish to a VSTS or TFS Package Management feed in this organization or collection, to NuGet.org, or to an internal NuGet repository.
- **Target feed:** Select the feed you want to publish to.

To publish to a VSTS Package Management feed, set the identity under which the build runs as a **Contributor** on the feed. If your build pipeline is configured to run using the project collection scope, then give the **Project Collection Build Service** identity contributor access to the feed. If your build pipeline is configured to run using project scope, then the **Project Build Service** identity should be given contributor access to the feed. You can check the scope in which your builds run by opening the editor for your build pipeline, and looking at the **Options** tab. To learn more about permissions for Package Management feeds, see Secure and share packages using feed permissions .

To publish to a external NuGet feed, you must first create a service connection to point to that feed. You can do this by going to **Project settings**, then choosing **Service connections**, and then creating a **New service connection**. Select the **NuGet** option for the service connection. Fill in feed URL and the API key or token to connect to the feed.

Publish symbols for your packages

When you push packages to a Package Management feed, you can also publish symbols.

Q&A

Where can I learn more about the VSTS and TFS Package Management service

Package Management in VSTS and TFS

SCVMM deployment

7/17/2018 • 7 minutes to read • Edit Online

You can automatically provision new virtual machines in System Center Virtual Machine Manager (SCVMM) and deploy to those virtual machines after every successful build. Before this guidance, read the web quickstart.

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

SCVMM connection

You need to first configure how VSTS connects to SCVMM. You cannot use Microsoft-hosted agents to run SCVMM tasks since the VMM Console is not installed on hosted agents. You must set up a self-hosted build and release agent on the same network as your SCVMM server.

You need to first configure how TFS connects to SCVMM. You must have a build and release agent that can communicate with the SCVMM server.

1. Install the **Virtual Machine Manager** (VMM) console on the agent machine by following these instructions. Supported version: System Center 2012 R2 Virtual Machine Manager.
2. Install the **System Center Virtual Machine Manager (SCVMM)** extension from Visual Studio Marketplace into TFS or VSTS:
   - 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. Create an SCVMM service connection in your project:
   - In your VSTS or TFS project in your web browser, navigate to the project settings and select **Services**.
   - In the **Services** tab, choose **New service connection**, and select **SCVMM**.
   - In the **Add new SCVMM Connection** dialog, enter the values required to connect to the SCVMM Server:
     - **Connection Name**: Enter a user-friendly name for the service connection such as **MySCVMMServer**.
     - **SCVMM Server Name**: Enter the fully qualified domain name and port number of the SCVMM server, in the form **machine.domain.com:port**.
     - **Username** and **Password**: Enter the credentials required to connect to the vCenter Server. Username formats such as **username**, **domain\username**, **machine-name\username**, and **.\username** are supported. UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.

Create new virtual machines from a template, VHD, or stored VM

One of the common actions that you can perform with every build is to create a new virtual machine to deploy the

build to. You use the SCMVMM task from the extension to do this and configure the properties of the task as follows:

- **Display name**: The name for the task as it appears in the task list.
- **SCVMM Service Connection**: Select a SCVMM service connection you already defined, or create a new one.
- **Action**: Select **New Virtual Machine using Template/Stored VM/VHD**.
- **Create virtual machines from VM Templates**: Set this option if you want to use a template.
  - **Virtual machine names**: Enter the name of the virtual machine, or a list of the virtual machine names on separate lines. Example `FabrikamDevVM`
  - **VM template names**: Enter the name of the template, or a list of the template names on separate lines.
  - **Set computer name as defined in the VM template**: If not set, the computer name will be the same as the VM name.
- **Create virtual machines from stored VMs**: Set this option if you want to use a stored VM.
  - **Virtual machine names**: Enter the name of the virtual machine, or a list of the virtual machine names on separate lines. Example `FabrikamDevVM`
  - **Stored VMs**: Enter the name of the stored VM, or a list of the VMs on separate lines in the same order as the virtual machine names.
- **Create virtual machines from VHD**: Set this option if you want to use a stored VM.
  - **Virtual machine names**: Enter the name of the virtual machine, or a list of the virtual machine names on separate lines. Example `FabrikamDevVM`
  - **VHDs**: Enter the name of the VHD or VHDX, or a list of names on separate lines in the same order as the virtual machine names.
  - **CPU count**: Specify the number of processor cores required for the virtual machines.
  - **Memory**: Specify the memory in MB required for the virtual machines.
- **Clear existing network adapters**: Set this option if you want to remove the network adapters and specify new ones in the **Network Virtualization** options.
- **Deploy the VMs to**: Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
- **Host Name** or **Cloud Name**: Depending on the previous selection, enter either a cloud name or a host machine name.
- **Placement path for VM**: If you selected **Host** as the deployment target, enter the path to be used during virtual machine placement. Example `C:\ProgramData\Microsoft\Windows\Hyper-V`
- **Additional Arguments**: Enter any arguments to pass to the virtual machine creation template. Example `-StartVM -StartAction NeverAutoTurnOnVM -StopAction SaveVM`
- **Wait Time**: The time to wait for the virtual machine to reach ready state.
- **Network Virtualization**: Set this option to enable network virtualization for your virtual machines. For more information, see Create a virtual network isolated environment.
- **Show minimal logs**: Set this option if you don't want to create detailed live logs about the VM provisioning process.

Delete virtual machines

After validating your build, you would want to delete the virtual machines that you created. You use the SCMVMM task from the extension to do this and configure the properties of the task as follows:

- **Display name**: The name for the task as it appears in the task list.
- **SCVMM Service Connection**: Select a SCVMM service connection you already defined, or create a new one.
- **Action**: Select **New Virtual Machine using Template/Stored VM/VHD**.
- **VM Names**: Enter the name of the virtual machine, or a comma-separated list of the virtual machine names. Example `FabrikamDevVM,FabrikamTestVM`
- **Select VMs From**: Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.

- **Host Name** or **Cloud Name**: Depending on the previous selection, enter either a cloud name or a host machine name.

Start and stop virtual machines

You can start a virtual machine prior to deploying a build, and then stop the virtual machine after running tests. Use the SCVMM task as follows in order to achieve this:

- **Display name**: The name for the task as it appears in the task list.
- **SCVMM Service Connection**: Select a SCVMM service connection you already defined, or create a new one.
- **Action**: Select **Start Virtual Machine** or **Stop Virtual Machine**.
- **VM Names**: Enter the name of the virtual machine, or a comma-separated list of the virtual machine names. Example `FabrikamDevVM,FabrikamTestVM`
- **Select VMs From**: Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
- **Host Name** or **Cloud Name**: Depending on the previous selection, enter either a cloud name or a host machine name.
- **Wait Time**: The time to wait for the virtual machine to reach ready state.

Create, restore, and delete checkpoints

A quick alternative to bringing up a virtual machine in desired state prior to running tests is to restore it to a known checkpoint. Use the SCVMM task as follows in order to do this:

- **Display name**: The name for the task as it appears in the task list.
- **SCVMM Service Connection**: Select a SCVMM service connection you already defined, or create a new one.
- **Action**: Select one of the checkpoint actions **Create Checkpoint**, **Restore Checkpoint**, or **Delete Checkpoint**.
- **VM Names**: Enter the name of the virtual machine, or a comma-separated list of the virtual machine names. Example `FabrikamDevVM,FabrikamTestVM`
- **Checkpoint Name**: For the **Create Checkpoint** action, enter the name of the checkpoint that will be applied to the virtual machines. For the **Delete Checkpoint** or **Restore Checkpoint** action, enter the name of an existing checkpoint.
- **Description for Checkpoint**: Enter a description for the new checkpoint when creating it.
- **Select VMs From**: Choose either **Cloud** or **Host** to select the set of virtual machines to which the action will be applied.
- **Host Name** or **Cloud Name**: Depending on the previous selection, enter either a cloud name or a host machine name.

Run custom PowerShell scripts for SCVMM

For functionality that is not available through the in-built actions, you can run custom SCVMM PowerShell scripts using the task. The task helps you with setting up the connection with SCVMM using the credentials configured in the service connection, and then runs the script.

- **Display name**: The name for the task as it appears in the task list.
- **SCVMM Service Connection**: Select a SCVMM service connection you already defined, or create a new one.
- **Action**: Select **Run PowerShell Script for SCVMM**.
- **Script Type**: Select either **Script File Path** or **Inline Script**.
- **Script Path**: If you selected **Script File Path**, enter the path of the PowerShell script to execute. It must be a fully-qualified path, or a path relative to the default working directory.
- **Inline Script**: If you selected **Inline Script**, enter the PowerShell script lines to execute.
- **Script Arguments**: Enter any arguments to be passed to the PowerShell script. You can use either ordinal parameters or named parameters.
- **Working folder**: Specify the current working directory for the script when it runs. The default if not provided is the folder containing the script.

Deploying build to virtual machines

Once you have the virtual machines set up, deploying a build to those virtual machines is no different than deploying to any other machine. For instance, you can:

- Use the PowerShell on Target Machines task to run remote scripts on those machines using Windows Remote Management.
- Use Deployment groups to run scripts and other tasks on those machines using build and release agent.

See also

- Create a virtual network isolated environment for build-deploy-test scenarios

VMware deployment

7/17/2018 • 5 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can automatically provision virtual machines in a VMware environment and deploy to those virtual machines after every successful build. Before this guidance, read the web quickstart.

VMware connection

You need to first configure how VSTS connects to vCenter. You cannot use Microsoft-hosted agents to run VMware tasks since the vSphere SDK is not installed on these machines. You have to a set up a self-hosted agent that can communicate with the vCenter server.

You need to first configure how TFS connects to vCenter. You have to a set up a self-hosted agent that can communicate with the vCenter server.

1. Install the VMware vSphere Management SDK to call VMware API functions that access vSphere web services. To install and configure the SDK on the agent machine:
   - Download and install the latest version of the Java Runtime Environment from this location.
   - Go to this location and sign in with your existing credentials or register with the website. Then download the **vSphere 6.0 Management SDK**.
   - Create a directory for the vSphere Management SDK such as **C:\vSphereSDK**. Do not include spaces in the directory names to avoid issues with some of the batch and script files included in the SDK.
   - Unpack the vSphere Management SDK into the new folder you just created.
   - Add the full path and name of the precompiled VMware Java SDK file **vim25.jar** to the machine's CLASSPATH environment variable. If you used the path and name **C:\vSphereSDK** for the SDK files, as shown above, the full path will be:
     `C:\vSphereSDK\SDK\vsphere-ws\java\JAXWS\lib\vim25.jar`
2. Install the VMware extension from Visual Studio Marketplace into TFS or VSTS.
   - 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 connection in your project:
   - Open your VSTS or TFS project in your web browser. Choose the **Settings** icon in the menu bar and select **Services**.
   - In the **Services** tab, choose **New service connection**, and select **VMware vCenter Server**.
   - In the **Add new VMware vCenter Server Connection** dialog, enter the values required to connect to the vCenter Server:

- **Connection Name**: Enter a user-friendly name for the service connection such as **Fabrikam vCenter**.
- **vCenter Server URL**: Enter the URL of the vCenter server, in the form `https://machine.domain.com/`. Note that only **HTTPS** connections are supported.
- **Username** and **Password**: Enter the credentials required to connect to the vCenter Server. Username formats such as **username**, **domain\username**, **machine-name\username**, and **.\username** are supported. UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.

Managing VM snapshots

Use the **VMware Resource Deployment** task from the VMware extension and configure the properties as follows to take snapshot of virtual machines, or to revert or delete them:

- **VMware Service Connection**: Select the VMware vCenter Server connection you created earlier.
- **Action**: Select one of the actions: **Take Snapshot of Virtual Machines**, **Revert Snapshot of Virtual Machines**, or **Delete Snapshot of Virtual Machines**.
- **Virtual Machine Names**: Enter the names of one or more virtual machines. Separate multiple names with a comma; for example, `VM1,VM2,VM3`
- **Datacenter**: Enter the name of the datacenter where the virtual machines will be created.
- **Snapshot Name**: Enter the name of the snapshot. This snapshot must exist if you use the revert or delete action.
- **Host Name**: Depending on the option you selected for the compute resource type, enter the name of the host, cluster, or resource pool.
- **Datastore**: Enter the name of the datastore that will hold the virtual machines' configuration and disk files.
- **Description**: Optional. Enter a description for the **Take Snapshot of Virtual Machines** action, such as `$(Build.DefinitionName).$(Build.BuildNumber)`. This can be used to track the execution of the build or release that created the snapshot.
- **Skip Certificate Authority Check**: If the vCenter Server's certificate is self-signed, select this option to skip the validation of the certificate by a trusted certificate authority.

> To verify if a self-signed certificate is installed on the vCenter Server, open the VMware vSphere Web Client in your browser and check for a certificate error page. The vSphere Web Client URL will be of the form `https://machine.domain/vsphere-client/`. Good practice guidance for vCenter Server certificates can be found in the VMware Knowledge Base (article 2057223).

Provisioning virtual machines

To configure the **VMware Resource Deployment** task to provision a new virtual machine from a template, use these settings:

- **VMware Service Connection**: Select the VMware vCenter Server connection you created earlier.
- **Action**: `Deploy Virtual Machines using Template`
- **Template**: The name of the template that will be used to create the virtual machines. The template must exist in the location you enter for the **Datacenter** parameter.
- **Virtual Machine Names**: Enter the names of one or more virtual machines. Separate multiple names with

a comma; for example, `VM1,VM2,VM3`

- **Datacenter**: Enter the name of the datacenter where the virtual machines will be created.
- **Compute Resource Type**: Select the type of hosting for the virtual machines: `VMware ESXi Host`, `Cluster`, or `Resource Pool`
- **Host Name**: Depending on the option you selected for the compute resource type, enter the name of the host, cluster, or resource pool.
- **Datastore**: Enter the name of the datastore that will hold the virtual machines' configuration and disk files.
- **Description**: Optional. Enter a description to identify the deployment.
- **Skip Certificate Authority Check**: If the vCenter Server's certificate is self-signed, select this option to skip the validation of the certificate by a trusted certificate authority. See the note for the previous step to check for the presence of a self-signed certificate.

Deploying build to virtual machines

Once you have the virtual machines set up, deploying a build to those virtual machines is no different than deploying to any other machine. For instance, you can:

- Use the PowerShell on Target Machines task to run remote scripts on those machines using Windows Remote Management.
- Use Deployment groups to run scripts and other tasks on those machines using build and release agent.

Deploy to an Azure Web App for Containers

7/20/2018 • 4 minutes to read • Edit Online

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 Azure at https://portal.azure.com.
2. In the Azure Portal, choose **Create a resource**, **New**, **Web**, then choose **Web App for Containers**.
3. Enter a name for your new web app, and select or create a new Resource Group.

4. Choose **Configure container** and select **Azure Container Registry**. Use the drop-down lists to select the registry you created earlier, and the Docker image and tag that was generated by the build pipeline.

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

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

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

You can configure Web App for Containers to automatically configure deployment as part of the CI/CD 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 pipeline 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 pipeline

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

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

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

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

4. Open the **Tasks** tab and select the **Environment 1** item. Configure the linked 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 connection to manually set up the connection.

- **App type**: Select **Linux App**.
- **App service name**: Select the web app you created earlier from your subscription.

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

- **Image Source**: Leave this set to **Container Registry**.
- **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 `<organization name>/<code-repo-name>` or just the repository name.

> You can find these values in the Azure portal in the **Overview** and **Repositories** tabs for your container registry.

5. Save the release pipeline.

Create a release to deploy your app

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

1. Choose **+ Release** and select **Create 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

Deploy to a Windows Virtual Machine

7/9/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

We'll show you how to set up continuous deployment of your ASP.NET or Node 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, 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**.

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 **Targets** tab, verify that your VM is listed.

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 pipeline that's automatically linked to the build pipeline.

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

2. Select the **IIS Website Deployment** template and choose **Apply**.

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

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.

5. Open the **Tasks** tab and select the **IIS Deployment** phase. For the **Deployment Group**, select the deployment group you created earlier (such as *myIIS*).

6. Save the release pipeline.

Create a release to deploy your app

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

1. Choose **+ Release** and select **Create Release**.
2. Check that the build you just completed is selected 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 app and verify its contents.

Next steps

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

Build multiple branches

7/17/2018 • 8 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can build every commit and pull request to your Git repository using VSTS or TFS. In this tutorial, we will discuss additional considerations when building multiple branches in your Git repository. You will learn how to:

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

Prerequisites

- You need a Git repository in VSTS, TFS, or GitHub with your app. If you do not have one, we recommend importing the sample .NET Core app into your VSTS or TFS project, or forking it into your GitHub repository. Note that you must use VSTS to build a GitHub repository. You cannot use TFS.
- You also need a working build for your repository.

Set up a CI trigger for a topic branch

A common workflow with Git is to create temporary branches from your master branch. These branches are called topic or feature branches and help you isolate your work. In this workflow, you create a branch for a particular feature or bug fix. Eventually, you merge the code back to the master branch and delete the topic branch.

- Designer
- YAML

Follow the steps below to create a CI trigger that will run a build for feature branches.

1. Select **Build and Release**, and then choose **Builds**.
2. Locate the build pipeline that services your master branch. 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 branch. Note that the filtered list of branches may not populate as you type `*`. You can still press **Enter** and save the branch filter.
6. Select the **Save & queue** menu and then Select **Save**.

Automatically build a change in topic branch

Your are now ready for CI for both the master branch and future feature branches that match the branch pattern. Every code change for the branch will use an automated build 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 your code in VSTS, TFS, or GitHub.
2. Create a new branch for your code that starts with `features/`, e.g., `features/feature-123`.
3. Make a change to your code in the feature branch and commit the change.
4. Navigate to the **Build and Release** menu in VSTS or TFS and select **Builds**.
5. Select the build pipeline for this repo. You should now see a new build executing for the topic branch. This build was initiated by the trigger you created earlier. Wait for the build to finish.

Your typical development process includes developing code locally and periodically pushing to your remote topic branch. Each push you make results in a build 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 or TFS so that certain tasks only execute on your master branch during a build run. You can use a single build with multiple branches and skip or perform certain tasks based on conditions.

- Designer
- YAML

1. Locate the build pipeline that services your master branch. Select **Edit**.
2. Choose a task in your build pipeline. If you are following the .NET Core sample, then select the **Publish Artifact** task.
3. Select **Control Options** for the task on the bottom right hand part of your screen.
4. Select the dropdown for **Run this task** and choose **Custom conditions**.

5. Enter the following snippet:

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

6. Select **Save & queue**.

7. Choose your **topic branch**. Select **Queue**. We are not building the master branch, and the task for **Publish artifacts** will not execute.
8. Select the build to monitor the progress. Once the build completes, confirm the build skipped the **Publish artifacts** task.

Validate pull requests

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

GitHub repository

- Designer
- YAML

1. Navigate to your project in VSTS or TFS. 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 build pipeline. A build will be queued or completed for the merge commit of your pull request.

VSTS or TFS repository

1. Navigate to the **Code** hub in VSTS or TFS.
2. Choose your **repository** and Select **Branches**. Choose the **master branch**.
3. You will implement a branch policy to protect the master branch. Select the **ellipsis** to the right of your branch name and Select **Branch policies**.
4. Choose the checkbox for **Protect this branch**. There are several options for protecting the branch.
5. Under the **Build validation** menu choose **Add build policy**.
6. Choose the appropriate build pipeline.
7. Ensure **Trigger** is set to automatic and the **Policy requirement** is set to required.
8. Enter a descriptive **Display name** to describe the policy.
9. Select **Save** to create and enable the policy. Select **Save changes** at the top left of your screen.
10. To test the policy navigate to the **Pull request** menu in VSTS or TFS.
11. Select **New pull request**. Ensure your topic branch is set to merge into your master branch. Select **create**.
12. Your screen displays the **policy** being executed.
13. Select the **policy name** to examine the build. If the build succeeds your code will be merged to master. If the build fails the merge is blocked.

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

Use retention policies to clean up your completed builds

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

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

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

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

Next steps

In this tutorial, you learned how to manage CI for multiple branches in your Git repositories using VSTS or TFS.

You learned how to:

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

Build your GitHub repository

7/19/2018 • 4 minutes 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
- Add a build status badge to your README file

Prerequisites

- Before you read this topic, you should understand the type of build you'd like to work with: designer or YAML.
- You need a GitHub repository with your app. If you do not have one, we recommend forking the sample .NET Core app into your own GitHub repo. You must have owner or admin privileges on the repository that you work with.

Set up CI for your GitHub repository

- Designer
- YAML

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 4 below.

1. Navigate to your VSTS organization and project. Select **Build and Release**, and then select **Builds**.
2. Select **New** to create a new build pipeline.
3. Select **GitHub** for the type of repository.
4. Give your connection a name, and then select the **Authorize using OAuth** button. Optionally you can use a GitHub **personal access token** instead of OAuth.
5. When prompted, sign in to your **GitHub account**. Then select **Authorize** to grant access to your VSTS organization. If you already are signed into GitHub in another browser tab, you may not see this step.
6. Choose the repository that contains the sample you forked earlier and select **Continue**.
7. Select the **ASP.NET Core** build template or a template that is appropriate for your application.
8. Choose **Hosted VS2017** for Agent queue.
9. Select **Triggers**. Enable **Continuous integration** for your builds. Ensure you include the `master` branch under **Branch filters**. This setting ensures each commit to `master` in GitHub will trigger a build via a GitHub webhook.
10. Select **Save & queue** to save your build pipeline and create the first build.
11. Once the build completes, select the name of the pipeline in the build results page to navigate to the history of builds for that pipeline. Take a note of the `definitionId` in the URL. You will need this to set up the build badge in upcoming steps.

Add a build status badge

This section explores possibilities for further integrating VSTS and GitHub. You will create a build badge for the VSTS build pipeline by populating a GitHub readme file with markdown that points to the build badge URL.

1. Navigate to your GitHub account. Select **Code**. Create a Readme.md file unless one already exists.
2. For this step, paste the following markdown into your Readme.md file. Replace the tokens using the name of the build pipeline and the `definitionId` that you obtained above.

```
[![Build status](https://{your-vsts-organization}.visualstudio.com/{your-vsts-project}/_apis/build/status/{build-definition-name}?branch=master)](https://{your-vsts-organization}.visualstudio.com/{your-vsts-project}/_build/latest?definitionId={definitionId}&branch=master)]
```

3. **Commit** your Readme.md file to the repository. The rendered Readme now shows the build badge.
4. Click on the build badge to navigate to the last completed build in VSTS.
5. In the build results page, select the commit id in the **Timeline** tab. This link navigates you directly to the GitHub commit.

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 pipelines by creating a GitHub service connection 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

Build multiple branches

Build OSS repositories

7/9/2018 • 3 minutes to read • Edit Online

VSTS Public Projects

If you manage an Open Source Software (OSS) project, then you can use VSTS public projects to automatically build and validate every pull request and commit to your repository.

This article provides common tasks for building OSS repositories.

Create a public project in VSTS

Your code may be in a GitHub public repository or in a VSTS public project.

- GitHub repo
- VSTS Git repo

Create a new VSTS public project using the steps described here. By configuring build and release pipelines in a VSTS public project, your builds and releases are visible to everyone. When users outside of your organization fork your repository and submit pull requests, they are able to view the status of builds that automatically validate those pull requests. They can also drill into the build logs and test results.

Create a build pipeline

- GitHub repo
- VSTS Git repo

Follow the steps in Build your GitHub repo to set up a build pipeline and to add a build badge to the `Readme.md` file at the root of your repository.

Validate contributions from forks

> **NOTE**
> This feature is currently unavailable for security hardening. It is expected to return in July 2018.

- GitHub repo
- VSTS Git repo

> **IMPORTANT**
> These settings impact the security of your build.

By default, your build pipeline is not automatically triggered for pull requests from forked repositories. You can change this behavior, but there are additional security considerations that you must take into account. To enable building pull requests from forks:

1. Navigate to your VSTS project. Select **Build and Release**, and then select **Builds**. Locate your build pipeline, and select **Edit**.
2. Select the **Triggers** tab. Enable the checkbox labeled **Build pull requests from forks of this repository**.

By default, secrets associated with your build pipeline 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 connection credentials
  - Files from the Secure Files library
  - Build variables marked **secret**

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 Microsoft-hosted agent to build pull requests from forks. Microsoft-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 self-hosted 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. Do not enable the checkbox labeled **Make secrets available to builds of forks**. Otherwise, secrets could leak during a build.

Other restrictions

You should be aware of the following restrictions when running builds in a VSTS public project.

1. **Cross-project access:** All builds in a public project run with a project scoped access token. In other words, builds in a public project can only access resources such as code, work items, or test results within the project and not from another project of the VSTS organization.
2. **Package management:** If your builds need access to packages from VSTS Package Management, then you must explicitly grant permission for the **Project Build Service** organization to access the package feeds.

Deploy a web app to Azure App Services

6/22/2018 • 3 minutes 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.

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 pipeline:

   - If you've just completed a CI build (see above), choose the link (for example, *Build 20170815.1*) to open the build summary. Then choose **Release** to start a new release pipeline that's automatically linked to the build pipeline.

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

2. The easiest way to create a release pipeline is to use a template. If you are deploying a Node 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 pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the **+ Add** link and select your build artifact.

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.

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

5. Open the **Tasks** tab and, with **Environment 1** selected, configure the task property variables as follows:

   - **Azure Subscription:** Select a connection from the list under **Available Azure Service Connections** or create a more restricted permissions connection to your Azure subscription. If you are using 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 connection to manually set up the connection.

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

> **NOTE**: Some settings for the tasks may have been automatically defined as environment variables when you created a release pipeline from a template. These settings cannot be modified in the task settings; instead you must select the parent environment item in order to edit these settings.

6. Save the release pipeline.

Create a release to deploy your app

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

1. Choose **+ Release** and select **Create 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 step

- Customize web app deployment

Define your multi-stage continuous deployment (CD) process

7/19/2018 • 9 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 pipeline 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 pipeline 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
  - Azure DevOps project
  - Deploy to IIS web server on Windows
- Two separate targets where you will deploy the app. These could be virtual machines, web servers, on-premises physical deployment groups, or other types of deployment target. In this example, we are using Azure App Services website instances. If you decide to do the same, you will have to choose names that are unique, but it's a good idea to include "QA" in the name of one, and "Production" in the name of the other so that you can easily identify them. Use the Azure portal to create a new web app.

Configure the triggers in your release pipeline

In this section, you will check that the triggers you need for continuous deployment are configured in your release pipeline.

1. In the **Build & Release** hub, open the **Releases** tab. Select your release pipeline 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.

For more information, see Release triggers in the Release Management documentation.

3. 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 pipeline.

Notice that you can also define artifact filters that determine a condition for the release to proceed, and set up a schedule for deployments. You can use features to, for example, specify a branch from which the build artifacts must have been created, or a specific time of day when you know the app will not be heavily used. For more information, see Environment triggers in the Release Management documentation.

Extend a release pipeline by adding environments

In this section, you will add a new environment to the release pipeline. 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.

1. In the **Pipeline** tab of your release pipeline, select the existing environment and rename it to **Production**.

2. Open the **+ Add** drop-down list and choose **Clone environment** (the clone option is available only when an existing environment is selected).

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.

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.

7. Recall that this environment is a clone of the original **Production** environment in the release pipeline. 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.

If you are using a different type of task to deploy your app, the way you change the target for the deployment may differ. For example, if you are using deployment groups, you may be able to select a different deployment group, or a different set of tags within the same deployment group.

> **NOTE**: Some settings for the tasks may have been automatically defined as environment variables when you created a release pipeline from a template. These settings cannot be modified in the task settings; instead you must select the parent environment item in order to edit these settings.

Add approvals within a release pipeline

The release pipeline you have modified deploys to test and then to production. If the deployment to test fails, the trigger on the production 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 pipeline to achieve this.

1. Back in the **Pipeline** tab of the release pipeline, 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.

2. In the **Approvers** section, choose your organization 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.

You can add as many approvers as you need, both individual accounts and organization 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 pipeline.

Create a release

Now that you have completed the modifications to the release pipeline, it's time to start the deployment. To do this, you create a release from the release pipeline. A release may be created automatically; for example, the continuous deployment trigger is set in the release pipeline. This means that modifying the source code will start a new build and, from that, a new release. However, in this section you will create a new release manually.

1. Open the **Release** drop-down list and choose **Create release**.

2. Enter a description for the release, check that the correct artifacts are selected, and then choose **Create**.

3. After a few moments, a banner appears indicating that the new release was created. Choose the link (the name of the release).

4. The release summary page opens showing details of the release. In the **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.

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.

5. Choose the **Approve or Reject** link to open the approval dialog. Enter a brief note about the approval, and choose **Approve**.

Notice that you can defer a deployment to a specific day and time; for example, a time when you expect the app to be only lightly loaded. You can also reassign the approval to another user. Release administrators can open *any* approval and over-ride it to accept or reject that deployment.

Monitor and track deployments

In this section, you will see how you can monitor and track deployments - in this example to two Azure App Services websites - from the release you created in the previous section.

1. In the release summary 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.

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.

2. 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.

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.

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.

5. Open the deployed production app in your browse. For example, for an Azure App Services website, from the URL `http://[your-app-name].azurewebsites.net`

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

7/17/2018 • 5 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

By using a combination of manual deployment approvals, gates, and manual intervention within a release pipeline in 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 for your DevOps CI/CD processes.

In this tutorial, you learn about:

- Extending the approval process with gates
- Extending the approval process with manual intervention
- Viewing and monitoring approvals and gates

Prerequisites

This tutorial extends the tutorial Define your multi-stage continuous deployment (CD) process. **You must have completed that tutorial first.**

You'll also need a **work item query** that returns some work items from VSTS or TFS. This query is used in the gate you will configure. You can use one of the built-in queries, or create a new one just for this gate to use. For more information, see Create managed queries with the query editor.

In the previous tutorial, you saw a simple use of manual approvals to allow an administrator to confirm that a release is ready to deploy to the production 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 or criteria, without requiring user intervention.

1. In the **Releases** tab of the **Build & Release** hub, select your release pipeline and choose **Edit** to open the pipeline editor.

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.

3. To allow gate functions to initialize and stabilize (it may take some time for them to begin returning accurate results), you configure a delay before the results are evaluated and used to determine if the deployment should be approved or rejected. For this example, so that you can see a result reasonably quickly, set the delay to a short period such as one minute.

4. Choose **+ Add** and select the **Query Work Items** gate.

5. 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).

You'll need to open the **Advanced** section to see the **Maximum Threshold** setting. For more details about the gate arguments, see Work Item Query task.

6. Open the **Options for all gates** section and specify the timeout and the sampling interval. For this example, choose short periods so that you can see the results reasonably quickly. The minimum values you can specify are 6 minutes timeout and 5 minutes sampling interval.

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 you release pipeline.

For more information about using other types of approval gates, see Approvals and gates.

Configure a manual intervention

Sometimes, you may need to introduce manual intervention into a release pipeline. For example, there may be tasks that cannot be accomplished automatically such as confirming network conditions are appropriate, or that specific hardware or software is in place, before you approve a deployment. You can do this by using the **Manual Intervention** task in your pipeline.

1. In the release pipeline editor, open the **Tasks** editor for the **QA** environment.

2. Choose the ellipses (**...**) in the **QA** deployment process bar and then choose **Add agentless phase**.

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.

4. Configure the task by entering a message (the **Instructions**) to display when it executes and pauses the release process.

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 pipeline and start a new release.

View the logs for approvals

You typically need to validate and audit a release and the associated deployments after it has completed, or even during the deployment 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.

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.

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.

4. 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.

Altogether, by using a combination of manual approvals, approval gates, and the manual intervention task, you've seen how can configure a release pipeline with all the control and auditing capabilities you may require.

Next step

Deploy to IIS web servers on Windows

Tutorial: Integrate your Jenkins CI jobs with VSTS CD to Azure

7/19/2018 • 12 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Visual Studio Team Services (VSTS) provides integration with Jenkins so that you can use Jenkins for Continuous Integration (CI) while gaining several DevOps benefits from a VSTS release pipeline to Azure such as:

- Reusing your existing investments in Jenkins Build jobs
- Tracking work items and related code changes
- View the entire CI/CD pipeline from a central location
- Deploy various Azure services consistently via VSTS
- Enforce quality of builds to gate deployments
- Define work flows such as manual approval processes and CI triggers

In this tutorial, you use Jenkins to build a **Java web app** that uses Maven and VSTS or Team Foundation Server (TFS) to deploy to an **Azure App Service**.

You will:

- Get the Sample App
- Configure Jenkins credentials and the VSTS plugin
- Configure a Jenkins CI build with VSTS integration
- Create a Jenkins service connection and Service Hooks in VSTS
- Create a VSTS Release Pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- (Optionally) Create a VSTS build pipeline to wrap the Jenkins CI job

Before you begin

- An Azure subscription. You can get one free through Visual Studio Dev Essentials.
- A Tomcat and Java 8 based Azure Web App. You can follow the steps for creating one here
- Basic Jenkins and Git knowledge.
- You need access to a Jenkins server with Maven configured. If you have not yet created a Jenkins server, see Create a Jenkins master on an Azure Virtual Machine.
- Sign in to your VSTS organization (`https://{youraccount}.visualstudio.com`). You can get a free VSTS organization.

  > **NOTE**
  > For more information, see Connect to VSTS.

Get the sample app

You need an app stored in a Git repository. You use this app to build and deploy. For this tutorial, we recommend you use this sample Java app available from GitHub. This tutorial uses a Java and Maven sample application that is configured for deployment to Azure App Service. If you want to work with your own repository, you should configure a similar sample.

1. In VSTS, on the **Code** hub for your VSTS project, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the following URL into the **Clone URL** text box.

   ```
   https://github.com/Azure-Samples/app-service-maven
   ```

3. Click **Import** to copy the sample code into your Git repo.
4. Select **Clone** at the top right, and keep the **clone url** for future steps in this tutorial.

Configure Jenkins credentials and the VSTS plugin

You must configure credentials for connecting to VSTS, and a plug in for **VS Team Services Continuous Deployment** on your Jenkins server. When using credentials to connect to VSTS, it is a best practice to use a **personal access token (PAT)**. You also need to create a Jenkins credential to use in your Jenkins build jobs.

> **NOTE**
> Ensure the PAT you use for the following steps contains the **Release (read, write, execute and manage), Code (read), Build (read and execute) permissions in VSTS**.

1. Create a PAT in your VSTS account. Jenkins requires this information to access your VSTS organization. Ensure you **store** the token information for upcoming steps in this section. Read How do I create a personal access token for VSTS and TFS to learn how to generate a PAT, or use an existing PAT if you have one.
2. Open your Jenkins account and select **Credentials**, **System**, and then choose **Add Domain**.
3. Enter a **name**, **description**, and then select **Add hostname**.
4. In the **Include** text box, enter `\*.visualstudio.com` to enable this credential for all VSTS organizations. You can optionally enter your specific VSTS organization.
5. Select **Ok**. You will see a message about empty credentials. Select the link for **adding some credentials**.
6. For **Kind**, choose **Username with password**. Enter the **Username** and the **PAT** you created earlier for **Password**. Select **Ok**.
7. **Navigate** back to the **Jenkins** home page.
8. Choose **Manage Jenkins**.
9. In the **Manage Jenkins** page, choose **Manage Plugins**.
10. Filter the **available list** to find the **VS Team Services Continuous Deployment** plug in and choose the **Install without restart** option. You can find additional information about the **VSTS plugin** here.

Configure a Jenkins CI build with VSTS integration

You create a Jenkins build job to use the source code stored in your VSTS repository and execute a basic build. Ensure your Jenkins server has **Maven** installed and configured. You also create triggers to enable CI builds when

code changes are pushed to the repository, and a CD trigger to initiate a VSTS release after the Jenkins build completes via a post build action.

1. Navigate to your Jenkins server. Click **New Item**. Enter an **item name**.
2. Choose **Freestyle project**. Select **OK**.
3. In the **Source Code Management** tab, select **Git** and enter the **clone url** you saved earlier for the VSTS **Repository URL** and the branch containing your app code. If you are using Team Foundation Server, you can choose the option for **Team Foundation Version Control (TFVC)**.

4. Select the **Credentials** drop down and choose the credential you created earlier. You should successfully authenticate to your VSTS repository and not receive errors before continuing. If you see errors, you likely have an issue with your credentials and VSTS **PAT**.
5. Select the **Build Triggers** tab, then **tick** the check boxes for **Build when a change is pushed to TFS/Team Services** and **Build when a change is pushed to a TFS pull request**. These two options rely on **VSTS Service Hooks** which you create in the next section.
6. Select the **Build** tab, **Add build step**, and then choose **Invoke top-level Maven targets**.
7. Select **Advanced...**. Enter **clean package** for **Goals**, and then enter **pom.xml** for **POM**.
8. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Archive the artifacts**.
9. For **Files to archive** enter the following:

```
target/*.war
```

10. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Trigger release in TFS/Team Services**.
11. Enter a **collection URL**. An example is `http://{YourVSTSAccount}.visualstudio.com/DefaultCollection`
12. Enter the **project** and choose a **release pipeline** name. Store the **release pipeline** name since it is needed

in later steps of this tutorial.

13. Enter the **username** and **PAT** you created earlier.

14. **Save** the Jenkins project.

Create a Jenkins service connection and Service Hooks in VSTS

You configure a Jenkins service connection to allow VSTS to connect to your Jenkins server. You must also configure two Jenkins service hooks so you can execute CI builds via automated triggers for both simple commits as well as pull requests to your VSTS Git repository.

1. Open the **Services** page in VSTS, open the **New service connection** list, and choose **Jenkins**.

2. Enter a name for the connection.

3. Enter the URL of your Jenkins server, and if using **http** tick the **Accept untrusted SSL certificates** option. An example URL is: `http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`

4. Enter the **user name and password** for your Jenkins account.

5. Choose **Verify connection** to check that the information is correct.

6. Choose **OK** to create the service connection.

7. From the **Service Hooks** page in VSTS, Select the **+** to add a new service and choose **Jenkins**. Select **next**.

8. Choose **Code pushed** for the type of event trigger. Select your **code repository** that contains the sample application you imported earlier. Select **Next**.

9. Enter the URL of your Jenkins server. An example is below.
`http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`

10. Enter the **user name and password** for your Jenkins account.

11. Select the Jenkins build you created earlier.

12. Choose **Test** to check that the information is correct.

13. Choose **Finish** to create the service connection.

14. Repeat the steps in this section to create another **service hook** for the **pull request merge attempted** trigger type. This will allow either simple commits to the repository as well as pull requests to both trigger the Jenkins build.

Create a VSTS release pipeline for CD to Azure

A release pipeline specifies the process VSTS executes to deploy the app. In this example, you deploy your app that originates from the Jenkins CI system. You deploy to an Azure App Service running Tomcat and Java.

To create the release pipeline in VSTS:

1. Open **Releases** on the **Build & Release** hub in VSTS, and choose **Create release pipeline**.
2. Select the **Empty** template by choosing **Start with an Empty process**.
3. In the **Artifacts** section, click on **+ Add Artifact** and choose **Jenkins** for **Source type**. Select your Jenkins service connection. Then select the Jenkins source job and choose **Add**.
4. Next to the **1 phase, 0 environments** link, select the **+** on the **Agent Phase** to add a task to the phase.
5. Search for the **Azure App Service Deploy** task. Select **Add** to add the task.
6. Choose your **Azure Subscription**. If you do not have an existing Azure connection in VSTS, you can follow the steps here to create one.
7. Enter a name for your existing Web App for the **App Service name**. This **name** must match your existing **Web App name** from the prerequisites.
8. For the **Package or folder** setting, select the ellipsis to browse, and then select your **.war artifact**.
9. Ensure the **name** for your release pipeline matches the same name you chose earlier during the **Create a Jenkins service connection and Service Hooks in VSTS** steps.
10. Click **Save**, and then click **OK** to save the release pipeline.

Test the CI/CD pipeline with a pull request

You can initiate the CI build and the subsequent CD deployment to Azure by completing a pull request into your master branch. The Jenkins build will initiate due to the service hook you set up earlier, and the Jenkins post build action will initiate a VSTS release which will deploy your app to the Azure App Service.

1. Navigate to **Code** in VSTS, then select your **repository**.
2. Select **New branch** to create a branch from the master branch. Enter a **name** for your new branch, and then select **Create branch**.
3. Select your new branch, then navigate to the **src/main/webapp/index.jsp** file.
4. Select **Edit**, then make a change to the **Title** for the **index.jsp page**, and then **Commit** your changes.
5. Select **Pull Requests**, then select **New Pull Request**. Select **Create** to issue a pull request from your branch to master.
6. Select **Approve**, then select **Complete**, then **Complete Merge**. This code change will initiate a CI build in Jenkins.
7. Navigate to your **Jenkins dashboard** and examine the build that is executing. Once it finishes, you can navigate to VSTS to watch the **Release Pipeline** execute. In a few moments your latest changes will be deployed to the **Azure App service.**

You are now using Jenkins CI builds with a VSTS code repository and release pipeline to perform CI/CD to Azure App Service. You can easily track your code changes and deployments via the rich reporting capabilities of VSTS, and leverage Jenkins to execute CI builds.

(Optionally) Create a VSTS build pipeline to wrap the Jenkins CI job

There is an additional approach (pattern) possible when integrating Jenkins and VSTS CI/CD. You can optionally

create a VSTS build pipeline to wrap the Jenkins build job, which will help you monitor the Jenkins job from VSTS. This pattern works slightly differently by using the built-in VSTS build capabilities to trigger the pipeline when new code is pushed to the repository. The approach for using both VSTS build and Jenkins together requires both VSTS build and Jenkins agents. To enable this pattern, you will modify a few items for the VSTS and Jenkins CI/CD pipeline that you created in earlier steps.

1. Navigate to the Jenkins build job you created earlier.
2. Select the **Build Triggers** tab, then remove the **tick** for the check boxes for **Build when a change is pushed to TFS/Team Services** and **Build when a change is pushed to a TFS pull request**. These two options are not needed for this pattern since the VSTS build pipeline has the ability to trigger the CI/CD pipeline when new code is pushed to the repo.
3. Navigate to your **VSTS organization**.
4. Open **Builds** on the **Build & Release** hub in VSTS, and choose **+ New** to create a new build pipeline.
5. Ensure **VSTS Git** is selected for the source, and then select **Continue**.
6. Search for the **Jenkins** template, and then select **Apply**.
7. Select **Process**, and then select **Hosted VS 2017** for the **Agent queue**.
8. For the **Job name** parameter, enter the **Jenkins Job name** you created in the earlier steps of this tutorial. This name must match exactly. This will allow you to queue the Jenkins job and download the artifacts it produces.
9. Select the **Jenkins service connection**.
10. Select **Get sources**, and then check the **Don't sync sources** option. Since the Jenkins job is handling fetching the sources from VSTS, we do not need the VSTS build pipeline to perform this step. However, this step will configure a local working directory on the VSTS build agent.
11. Ensure the **Report build status** option is enabled so the build status will be reported on the code tab for the VSTS repository.
12. Select the **Jenkins** build tasks under **Phase 1**, and then examine the additional options provided by the tasks. There is a task to queue the Jenkins job, download the artifacts produced by the job, and then finally to publish the artifacts which can later be used by the VSTS release pipeline.
13. Select the **Queue Jenkins Job** task. **Capture console output and wait for completion** is enabled to help provide the Jenkins logging information to VSTS, and also it will cause the VSTS build to wait for this step to complete successfully in Jenkins before proceeding. This causes the VSTS build to succeed for fail based on the Jenkins result. **Capture pipeline output and wait for completion** is similarly enabled to ensure VSTS succeeds or fails based on the entire Jenkins pipeline result. These two options help you view Jenkins logging information in a single place (VSTS) for the entire CI/CD pipeline.
14. Open **Releases** on the **Build & Release** hub in VSTS, and choose the **release pipeline** you created earlier. Select the **ellipsis** and choose **Edit**.
15. **Delete** the artifact you used earlier. **Add** a new artifact by choosing your VSTS build pipeline. The VSTS build pipeline will now provide the artifact for the CD part of the pipeline.
16. In the **Artifacts** section, click on **+ Add Artifact** and choose **Jenkins** for **Source type**. Select your Jenkins service connection. Then select the Jenkins source job and choose **Add**.

You can now test the pipeline (as you did on earlier steps) with a pull request or by pushing new code to your branch. A VSTS build will initiate, a Jenkins job executes, and a VSTS release pipeline will deploy your changes to Azure.

Next Steps

In this tutorial, you automated the deployment of an app to Azure using Jenkins for the CI build and VSTS for release. You learned how to:

- Get the Sample App
- Configure Jenkins credentials and VSTS plugin
- Configure a Jenkins CI build with VSTS integration
- Create a Jenkins service connection and service hooks in VSTS
- Create a VSTS Release Pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- (Optionally) Create a VSTS build pipeline to wrap the Jenkins CI job

Improve code quality with branch policies

Tutorial: Deploy to Kubernetes on Azure Container Service (AKS) with Jenkins CI and VSTS CD

7/19/2018 • 16 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Visual Studio Team Services (VSTS) provides integration with Jenkins so that you can:

- Continue to use your existing investments in Jenkins.
- Create a single release process using VSTS that deploys to a variety of Azure targets.

In this tutorial, you use Jenkins for Continuous Integration (CI) and VSTS for Continuous Delivery (CD) to deploy a **Spring Boot app** to an **Azure Container Service (AKS) Kubernetes cluster**.

You will:

- Get the sample app
- Configure the sample app to build and push a Docker image to your ACR
- Configure the sample app to include a YAML file used for deploying to your AKS cluster
- Configure Jenkins credentials to connect to VSTS
- Configure Jenkins Maven global settings
- Create Jenkins Service Hooks in VSTS
- Configure a Jenkins CI build with VSTS integration
- Install the Release Management Utilty tasks VSTS extension
- Create a VSTS release pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- View the deployed sample app
- Delete your AKS cluster

Prerequisites

- A VSTS organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- An Azure subscription. You can get one free from Visual Studio Dev Essentials.
- The Azure Command-Line Interface (CLI).
- You need a Spring Boot app. You can fork the sample app found here.
- You need An Azure Container Registry (ACR). You can follow steps to deploy an ACR and login to the registry using the Azure CLI via the steps here.
- An AKS cluster. You can follow the steps for creating this here.
- Access to a Jenkins server with Maven and the VSTS plugin configured. If you have not yet created a Jenkins server, see Create a Jenkins master on an Azure Virtual Machine. Also, the following Jenkins plugins

must be installed:

- **VS Team Services Continous Deployment** plugin. You can find additional information about the **VSTS plugin** here.
- **Config File Provider** plugin. You can find additional information about the **Config File plugin** here.
- **Maven Integration** plugin. You can find additional information about the **Maven Integration plugin** here.

Get the sample app

You need an app stored in a Git repository. You use this app to build and deploy. For this tutorial, we recommend you use this Spring Boot sample app available from GitHub. This tutorial uses a Spring Boot sample application that is configured for deployment to an AKS cluster. If you want to work with your own repository, you should configure a similar sample.

1. In VSTS, on the **Code** hub for your VSTS project, select the option to **Import repository**.
2. In the **Import a Git repository** dialog box, paste the following URL into the **Clone URL** text box.

   ```
   https://github.com/spring-guides/gs-spring-boot-docker
   ```

3. Click **Import** to copy the sample code into your Git repo.
4. Select **Clone** at the top right, and keep the **clone URL** for future steps in this tutorial.

Configure the sample app to build and push a Docker image to your ACR

The Spotify Dockerfile Maven plugin is used to build and push a Docker image to a registry, such as ACR. The Spring Boot sample app's Maven **pom.xml** file is already configured to use this plugin.

You will find the **pom.xml** file in your repository in the folder named **complete**. You must update the **pom.xml** file with your ACR's login server:

1. Navigate to your VSTS project. Select the **Code** tab.
2. In the code explorer, expand the **complete** folder and select the **pom.xml** file.
3. In the **Contents** pane that shows the contents of the **pom.xml** file, select the **Edit** button.
4. Update the `<properties>` collection in the **pom.xml** with the login server value for your ACR. The login server is the name of your ACR appended with *.azurecr.io*. For example, **yourACRName.azurecr.io**.

   ```
   <properties>
      <docker.image.prefix>wingtiptoysregistry.azurecr.io</docker.image.prefix>
      <java.version>1.8</java.version>
   </properties>
   ```

5. Update the **com.spotify** plugin in the **pom.xml** file so that the `<configuration>` section includes the tag for the Docker image that will be built and pushed to your ACR. The tag will be set to the current Jenkins build id. Also, add the **useMavenSettingsForAuth** setting to the configuration so that in a later step, you can configure Maven to authenticate with your ACR which is a private registry.

```
<plugin>
  <groupId>com.spotify</groupId>
  <artifactId>dockerfile-maven-plugin</artifactId>
  <version>1.3.4</version>
    <configuration>
        <repository>${docker.image.prefix}/${project.artifactId}</repository>
        <tag>${env.BUILD_NUMBER}</tag>
        <useMavenSettingsForAuth>true</useMavenSettingsForAuth>
                <buildArgs>
                    <JAR_FILE>target/${project.build.finalName}.jar</JAR_FILE>
                </buildArgs>
   </configuration>
</plugin>
```

6. Select **Commit** and choose to commit these changes to the **master** branch.

Configure the sample app to include a YAML file used for deploying to your AKS cluster

The YAML file contains deployment settings for pulling the docker image from the ACR. Update this file to include the name of your ACR:

1. Navigate to your VSTS project. Select the **Code** tab.

2. In the code explorer, select the **complete** folder's ellipsis button to open the context menu that shows more actions.

3. From the context menu, select **New**, **File**.

4. Name the file **K8sDeploy.yaml** and select **Create**.

5. Copy and paste the following contents into the **K8sDeploy.yaml** file. Ensure the image prefix `<yourACRName>` is replaced with the name of your ACR appended with *.azurecr.io*. Also, notice that the image is tagged with **Build.BuildId**. This is a token that will be automatically replaced during the VSTS release so that it's set to the current Jenkins build id. For example, if the current Jenkins build id is 5, the full name of an image that will be pulled during the VSTS release will look similar to: yourACRName.azurecr.io/gs-spring-boot-docker:5.

```
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  name: gs-spring-boot-docker
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: gs-spring-boot-docker
    spec:
      containers:
      - image: <yourACRName>.azurecr.io/gs-spring-boot-docker:__Build.BuildId__
        name: gs-spring-boot-docker
        imagePullPolicy: Always
        ports:
        - containerPort: 8080
      imagePullSecrets:
        - name: regsecret
---
apiVersion: v1
kind: Service
metadata:
  name: gs-spring-boot-docker
spec:
  ports:
  - protocol: TCP
    port: 80
    targetPort: 8080
  selector:
    app: gs-spring-boot-docker
  type: LoadBalancer
```

6. Select **Commit** and choose to commit these changes to the **master** branch.

Configure Jenkins credentials to connect to VSTS

You must configure credentials for connecting to VSTS. When using credentials to connect to VSTS, it is a best practice to use a **personal access token (PAT)**. You also need to create a Jenkins credential to use in your Jenkins build jobs.

> **NOTE**
> Ensure the PAT you use for the following steps contains the **Release (read, write, execute and manage), Code (read), Build (read and execute) permissions in VSTS**.

1. Create a PAT in your VSTS account. Jenkins requires this information to access your VSTS organization. Ensure you **store** the token information for upcoming steps in this section. Read How do I create a personal access token for VSTS and TFS to learn how to generate a PAT, or use an existing PAT if you have one.

2. Open your Jenkins account and select **Credentials**, **System**, and then choose **Add Domain**.

3. Enter a **name**, **description**, and then select **Add hostname**.

4. In the **Include** text box, enter `\*.visualstudio.com` to enable this credential for all VSTS organizations. You can optionally enter your specific VSTS organization.

5. Select **Ok**. You will see a message about empty credentials. Select the link for **adding some credentials**.

6. For **Kind**, choose **Username with password**. Enter the **Username** and the **PAT** you created earlier for **Password**. Select **Ok**.

Configure Jenkins Maven global settings

Since your ACR is a private registry, you must configure the user name and password for your ACR in Maven's global settings in Jenkins. This will authenticate Maven so that it can push images to your ACR during a build.

1. Retrieve the password for your ACR using the Azure CLI. Replace `<yourACRName>` with the name of your registry.

   ```
   az acr credential show --name <acrName> --query passwords[0]
   ```

2. Navigate to your Jenkins server. Select **Manage Jenkins** and then select the **Managed files** link.

3. To create the Maven **settings.xml** file the first time, select the **Add a new Config** link.

4. Select the **Maven settings.xml** radio button and select the **Submit** button.

5. Enter a **Name** for the file.

6. In the **Content** window, update the `<servers>` section of the contents to include the name of your ACR and its password. For example:

   ```
   <server>
     <id>yourACRName.azurecr.io</id>
     <username>yourACRName</username>
     <password>e1M3Y3Z9vuUtusVx12+GoQvuAV3wsQFZ</password>
   </server>
   ```

7. Select the **Submit** button to save the settings.

Configure a Jenkins CI build with VSTS integration

You create a Jenkins build job to use the source code stored in your VSTS repository and execute a basic build. Ensure your Jenkins server has **Maven** installed and configured. You also create triggers to enable CI builds when code changes are pushed to the repository, and a CD trigger to initiate a VSTS release after the Jenkins build completes via a post build action.

1. Navigate to your Jenkins server. Click **New Item**. Enter an **item name**.

2. Choose **Maven project**. Select **OK**.

3. In the **Source Code Management** tab, select **Git** and enter the **clone URL** you saved earlier for the VSTS **Repository URL** and the branch containing your app code. If you are using Team Foundation Server, you can choose the option for **Team Foundation Version Control (TFVC)**.

4. Select the **Credentials** drop down and choose the credential you created earlier. You should successfully authenticate to your VSTS repository and not receive errors before continuing. If you see errors, you likely have an issue with your credentials and VSTS **PAT**.

5. Select the **Build Triggers** tab, then **tick** the checkboxes for **Build when a change is pushed to TFS/Team Services** and **Build when a change is pushed to a TFS pull request**. These two options rely on **VSTS Service Hooks** which you create in the next section.

6. Select the **Build** tab, set the **Root POM** to the relative path to the sample app's **pom.xml**: **complete/pom.xml**.

7. Enter **clean package** for **Goals**.

8. Select the **Advanced...** button. Set the **Settings file** to **provided settings.xml** and set **Provided Settings** to the name of the Maven settings.xml file that you created earlier.

9. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Archive the artifacts**.

10. For **Files to archive** enter the relative path to the Kubernetes YAML deployment file: **complete/K8sDeploy.yaml**.

11. Select the **Post-build Actions** tab. Choose **Add post-build action**, then select **Trigger release in TFS/Team Services**.

12. Enter a **Collection URL**. An example is `http://{YourVSTSAccount}.visualstudio.com/DefaultCollection`

13. Enter the **project** and choose a **release pipeline** name. Store the **release pipeline** name since it is needed in later steps of this tutorial.

14. Enter the **username** and **PAT** you created earlier.

15. **Save** the Jenkins project.

Create a Jenkins and AKS service connection in VSTS

You configure a Jenkins service connection to allow VSTS to connect to your Jenkins server. You also need to configure an AKS service connection to allow VSTS to access your AKS cluster to configure deployment.

1. Open the **Services** page in VSTS, open the **New service connection** list, and choose **Jenkins**.

2. Enter a name for the connection.

3. Enter the URL of your Jenkins server, and if using **http** tick the **Accept untrusted SSL certificates** option. An example URL is: `http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`

4. Enter the **user name and password** for your Jenkins account.

5. Choose **Verify connection** to check that the information is correct.

6. Choose **OK** to create the service connection.

Add a second service connection for the AKS cluster.

1. Opening the **New service connection** list, and choose **Kubernetes**.

2. Enter a name for the connection.

3. Set the **Server URL** to the cluster's fully qualified domain name, such as **http://yourAKSCluster-yourResourceGroup-e65186-1e0d187e.hcp.eastus.azmk8s.io**. Retrieve the fully qualified domain name using the Azure CLI. Replace `<yourResourceGroup>` with the name of the resource group that contains the AKS cluster. Replace `<yourAKSCluster>` with the name of the AKS cluster.

```
az aks show --resource-group <yourResourceGroup> --name <yourAKSCluster> --query fqdn
```

4. Set the **KubeConfig** to the access credentials for the AKS cluster. Use the Azure CLI to get these credentials:

```
az aks get-credentials --resource-group <yourResourceGroup> --name <yourAKSCluster>
```

   This command will get the access credentials for the AKS cluster. Navigate to the **.kube** folder under your home directory, such as **C:\Users\YOUR_HOMEDIR.kube**. Copy the contents of the **config** file and paste it in the **Kubernetes Connection** window. Select **OK** to create the service connection.

Create Jenkins Service Hooks in VSTS

You must also configure two Jenkins service hooks so you can execute CI builds via automated triggers for both simple commits as well as pull requests to your VSTS Git repository.

1. From the **Service Hooks** page in VSTS, Select the **+** to add a new service and choose **Jenkins**. Select **next**.

2. Choose **Code pushed** for the type of event trigger. Select your **code repository** that contains the sample application you imported earlier. Select **Next**.
3. Enter the URL of your Jenkins server. An example is below.
   `http://{YourJenkinsURL}.westcentralus.cloudapp.azure.com`
4. Enter the **user name and password** for your Jenkins account.
5. Select the Jenkins build you created earlier.
6. Choose **Test** to check that the information is correct.
7. Choose **Finish** to create the service connection.
8. Repeat the steps in this section to create another **service hook** for the **pull request merge attempted** trigger type. This will allow either simple commits to the repository as well as pull requests to both trigger the Jenkins build.

Install the Release Management Utilty tasks VSTS extension

A release pipeline specifies the process VSTS executes to deploy the app. In this example, you deploy your app that originates from the Jenkins CI system. You deploy to a Docker image running Tomcat and a Spring Boot app to an AKS cluster.

Before you create the release pipeline, you need to install a VSTS extension that will be used to replace the **K8sDeploy.yaml** file's **Build.BuildId** token with the current Jenkins build id.

1. In your VSTS organization, on the top right-hand side of the browser, Select the **Browse Marketplace** menu item. (The icon appears as a shopping bag.)
2. Search for the **Release Management Utility Tasks** extension provided by **Microsoft DevLabs**. The extension includes the **Tokenizer** utility.
3. Select the **Get it free** button.
4. Select your VSTS organization and select the **Install** button to install the extension.
5. After the extension is installed, select the **Proceed to Organization** button and navigate back to your VSTS project.

Create a VSTS release pipeline for CD to Azure

1. Open the **Releases** on the **Build & Release** hub, and choose **Create release pipeline**.
2. Select the **Empty** template by choosing **Start with an Empty process**.
3. In the **Artifacts** section, click on **+ Add Artifact** and choose **Jenkins** for **Source type**. Select your Jenkins service connection. Then select the Jenkins source job and choose **Add**.

Add two tasks to the release pipeline. The first task updates the **K8sDeploy.yaml** file to pull the image tagged with the current Jenkins build id.

1. Next to the **1 phase, 0 environments** link, select the **+** on the **Agent Phase** to add a task to the phase.
2. Search for the **Tokenize with XPath/Regular expressions** task which was added with the extension that was installed in the previous step. Select **Add** to add the task.
3. Set the **Source filename** to the **K8sDeploy.yaml** that is archived by the Jenkins job during the build. This task automatically replaces the **Build.BuildId** token with the current Jenkins build id.

The second task deploys to the AKS cluster:

1. Select the **+** button to add a second task. Search for the **Deploy to Kubernetes** task. Select **Add** to add the task.
2. Set the **Kubernetes Service Connection** to the name of the service connection that you created for the AKS cluster.
3. Set the **Command** to **apply**.
4. Select the **Use Configuration files** check box and set the **Configuration File** to the **K8sDeploy.yml** file.
5. Expand the **Container Registry Details** section of the task.
6. Set **Container Registry type** to **Azure Container Registry**.
7. Set **Azure subscription** to your subscription. If you do not have an existing Azure connection in VSTS, you can follow the steps here to create one.
8. Set **Azure Container Registry** to the name of your ACR.
9. Set **Secret name** to the secret provided in the **K8sDeploy.yaml** file which is named **regsecret**. This is the name of an object in the AKS cluster that is used to store an authentication token. The cluster uses this token to authenticate to the ACR to pull images.
10. Ensure the **name** for your release pipeline matches the same name you chose earlier during the **Create a Jenkins service connection and service hooks in VSTS** steps.
11. Click **Save**, and then click **OK** to save the release pipeline.

Test the CI/CD pipeline with a pull request

You can initiate the CI build and the subsequent CD deployment to Azure by completing a pull request into your master branch. The Jenkins build will initiate due to the service hook you set up earlier, and the Jenkins post build action will initiate a VSTS release which will deploy your app to the Azure App Service.

1. Navigate to **Code** in VSTS, then select your **repository**.
2. Select **New branch** to create a branch from the master branch. Enter a **name** for your new branch, and then select **Create branch**.
3. Select your new branch, then navigate to the **complete/src/main/java/hello/Application.java** file.
4. Select **Edit**, then make a change to the message displayed in the **home()** method and **Commit** your changes.
5. Select **Pull Requests**, then select **New Pull Request**. Select **Create** to issue a pull request from your branch to master.
6. Select **Approve**, then select **Complete**, then **Complete Merge**. This code change will initiate a CI build in Jenkins.
7. Navigate to your **Jenkins dashboard** and examine the build that is executing. Once it finishes, a new Docker image will be pushed to your ACR that is tagged with the current Jenkins build id. You can then navigate to VSTS to watch the **Release Pipeline** execute. In a few moments, the Docker image for the current Jenkins build will be deployed to your AKS cluster.

You are now using Jenkins CI builds with a VSTS code repository and release pipeline to perform CI/CD to **Azure Container Services (AKS)**. You can easily track your code changes and deployments via the rich reporting capabilities of VSTS, and leverage Jenkins to execute CI builds.

View the deployed sample app

Once the app is deployed to the AKS cluster, you can query the external IP address using **kubectl**, the Kubernetes command-line client. You can learn how to install and connect **kubectl** to your AKS Cluster by following these steps.

1. Use the following command for querying the external IP address for the deployed app:

```
kubectl get svc -w
```

kubectl will display the internal and external IP addresses; for example:

```
NAME                    CLUSTER-IP   EXTERNAL-IP   PORT(S)        AGE
kubernetes              10.0.0.1     <none>        443/TCP        19h
gs-spring-boot-docker   10.0.242.8   13.65.196.3   80:31215/TCP   3m
```

1. Use the external IP address to open the sample app in a web browser.

Delete your AKS cluster

When your AKS cluster is no longer needed, you can use the `az group delete` command to remove the resource group, which will remove all of its related resources; for example:

```
az group delete --name <yourAKSCluster> --yes --now-wait
```

Next Steps

In this tutorial, you automated the deployment of an app to Azure using Jenkins build and VSTS for release. You learned how to:

- Get the sample app
- Configure the sample app to build and push a Docker image to your ACR
- Configure the sample app to include a YAML file used for deploying to your AKS cluster
- Configure Jenkins credentials to connect to VSTS
- Configure Jenkins Maven global settings
- Create Jenkins Service Hooks in VSTS
- Configure a Jenkins CI build with VSTS integration
- Install the Release Management Utilty tasks VSTS extension
- Create a VSTS release pipeline for CD to Azure
- Test the CI/CD pipeline with a pull request
- View the deployed sample app
- Delete your AKS cluster

Integrate your Jenkins CI jobs with VSTS DevOps Deploy to Kubernetes with Fabric8

Build and Release Agents

7/19/2018 • 12 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To build your code or deploy your software you need at least one agent. As you add more code and people, you'll eventually need more.

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

Microsoft-hosted agents

If your build and release pipelines are in VSTS, then you've got a convenient option to build and deploy using a **Microsoft-hosted agent**. When you use a Microsoft-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 self-hosted agent.

> **TIP**
> You can try a Microsoft-hosted agent for no charge. If your build or release doesn't succeed, the issues will be reported in the logs.

Learn more about Microsoft-hosted agents.

Self-hosted agents

An agent that you set up and manage on your own to run build and deployment jobs is a **self-hosted agent**. You can use self-hosted agents in VSTS or Team Foundation Server (TFS). Self-hosted agents give you more control to install dependent software needed for your builds and deployments.

> **TIP**
> Before you install a self-hosted agent you might want to see if a Microsoft-hosted agent pool will work for you. In many cases this is the simplest way to get going. Give it a try.

You can install the agent on Linux, macOS, or Windows machines. You can also install an agent on a Linux Docker container. See the following topics for additional information on installing a self-hosted agent:

- macOS agent
- Red Hat agent
- Ubuntu 14.04 agent
- Ubuntu 16.04 agent
- Windows agent v2

- macOS agent
- Red Hat agent
- Ubuntu 14.04 agent
- Ubuntu 16.04 agent
- Windows agent v1

After you've installed the agent on a machine, you can install any other software on that machine as required by your build or deployment jobs.

Concurrent jobs

You might need more concurrent jobs to use multiple Microsoft-hosted or self-hosted agents at the same time:

- Concurrent jobs in VSTS

Concurrent pipelines

You might need more concurrent pipelines to use multiple agents at the same time:

- 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 pipeline, or when you queue a build or deployment, you specify certain **demands** of the agent. The system sends the job only to agents that have capabilities matching the demands specified in the pipeline. As a result, agent capabilities allow you to direct builds and deployments to specific agents.

You can view the system capabilities of an agent, and manage its user capabilities by navigating to the **Agent pools** hub and selecting the **Capabilities** tab for the desired agent.

- VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
- TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
- TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
- TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

The TFS URL doesn't work for me. How can I get the correct URL?

> **TIP**
> After you install new software on a agent, you must restart the agent for the new capability to show up.

Communication

Communication with VSTS

Communication with TFS

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 pipeline. 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 pipelines, release pipelines, or variable groups are secured as they are exchanged with the agent.

Here is a common communication pattern between the agent and TFS.

- An agent pool administrator joins the agent to an agent pool, and the credentials of the service account (for Windows) or the saved user name and password (for Linux and macOS) are used to initiate communication with TFS. The agent uses these credentials to listen to the job queue.
- The agent does not use asymmetric key encryption while communicating with the server. However, you can use HTTPS to secure the communication between the agent and TFS.

Communication to deploy to target servers

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

If your on-premises environments do not have connectivity to a Microsoft-hosted agent pool (which is typically the case due to intermediate firewalls), you'll need to manually configure a self-hosted agent on on-premises computer(s). The agents must have connectivity to the target on-premises environments, and access to the Internet to connect to 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. The identity of agent pool administrator is needed only at the time of registration and is not persisted on the agent, and is not used in any subsequent communication between the agent and VSTS or TFS. In addition, you must be a local administrator on the server in order to configure the agent. 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. Also, as explained above, this PAT is used only at the time of registering the agent, and not for subsequent communication.

To use a PAT with TFS, your server must be configured with HTTPS. See Web site settings and security.

Integrated

Connect a Windows agent to TFS using the credentials of the signed-in user through a Windows authentication scheme such as NTLM or Kerberos.

To use this method of authentication, you must first configure your TFS server.

1. Sign into the machine where you are running TFS.

2. Start Internet Information Services (IIS) Manager. Select your TFS site and make sure Windows Authentication is enabled with a valid provider such as NTLM or Kerberos.

Negotiate

Connect to TFS as a user other than the signed-in user through a Windows authentication scheme such as NTLM or Kerberos.

To use this method of authentication, you must first configure your TFS server.

1. Log on to the machine where you are running TFS.

2. Start Internet Information Services (IIS) Manager. Select your TFS site and make sure Windows Authentication is enabled with the Negotiate provider and with another method such as NTLM or Kerberos.

Alternate

Connect to TFS using Basic authentication. To use this method you must first configure HTTPS on TFS.

To use this method of authentication, you must configure your TFS server as follows:

1. Log on to the machine where you are running TFS.
2. Configure basic authentication. See Using tfx against Team Foundation Server 2015 using Basic Authentication.

Interactive vs. service

You can run your agent as either a service or an interactive process. Whether you run an agent as a service or interactively, you can choose which account you use to run the agent. Note that this is different from the credentials that you use when you register the agent with 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 project collection or organization.

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_organization}.visualstudio.com/_admin/_AgentPool`
- TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
- TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
- TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

The TFS URL doesn't work for me. How can I get the correct URL?

Q & A

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

In many cases, yes. Specifically:

- If you use a self-hosted agent you can run incremental builds. For example, you define a CI

build process that does not clean the repo and does not perform a clean build, your builds will typically run faster. When you use a Microsoft-hosted agent, you don't get these benefits because the agent is destroyed after the build or release process is completed.

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

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

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

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

You might also run into problems if 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_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?

2. Click the pool 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

7/19/2018 • 7 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Instead of managing each agent individually, you organize agents into **agent pools**. An agent pool defines the sharing boundary for all agents in that pool. In TFS, pools are scoped across all of your Team Foundation Server (TFS); so you can share an agent pool across project collections and projects. In VSTS, agent pools are scoped to the VSTS organization; so you can share an agent pool across projects.

An **agent queue** provides access to an agent pool. When you create a build or release pipeline, you specify which queue it uses. Queues are scoped to your project in TFS 2017 and newer and in VSTS, so you can only use them across build and release pipelines within a project.

To share an agent pool with multiple projects, you create an agent queue pointing to that pool in each of those projects. While multiple queues across projects can use the same agent pool, multiple queues within a 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 project collections.

You create and manage pools from the Agent pools tab.

- VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
- TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
- TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
- TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

The TFS URL doesn't work for me. How can I get the correct URL?

You create and manage your queues from the Agent queues tab.

- VSTS: `https://{your_organization}.visualstudio.com/{project-name}/_admin/_AgentQueue`
- TFS 2017 and newer: `https://{your_server}/tfs/{collection-name}/{project-name}/_admin/_AgentQueue`
- TFS 2015 RTM: `http://{your_server}:8080/tfs/_admin/_buildQueue`
- TFS 2015.3: `http://{your_server}:8080/tfs/{collection-name}/_admin/_AgentQueue`

The TFS URL doesn't work for me. How can I get the correct URL?

Default agent pools

We provide the following agent pools by default:

- **Default** pool: Use it to register self-hosted agents that you've set up.
- **Hosted** pool (VSTS only): Contains at least one free Microsoft-hosted agent, and also any Microsoft-hosted agents you've purchased. The **Hosted** pool is the built-in pool that is a collection of Microsoft-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 Microsoft-hosted agents, see Microsoft-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 Microsoft-hosted agents.
- **Hosted Linux** pool (VSTS only): Enables you to build and release on Linux machines without having to configure a self-hosted agent. The agents in this pool run on an Ubuntu Linux host inside the **vsts-agent-docker** container.
- **Hosted macOS** pool (VSTS only): Enables you to build and release on Mac machines without having to configure a self-hosted agent. This option affects where your data is stored. Learn more

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

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.
Service Account	Members of this role can use the pool to create an agent queue in a 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 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 pipelines.
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 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 organization-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 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 organization 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 organizations 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.

Microsoft-hosted agents

7/19/2018 • 4 minutes to read • Edit Online

If your build and release pipelines are in VSTS, then you've got a convenient option to build and deploy using a **Microsoft-hosted agent**. When you use a Microsoft-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 self-hosted agent.

> **TIP**
> You can try a Microsoft-hosted agent for no charge. If your build or release doesn't succeed, the issues will be reported in the logs.

Use a Microsoft-hosted agent

To use a Microsoft-hosted agent pool, first decide which queue to use:

IF YOUR DEVELOPMENT TEAM USES...	...THEN CHOOSE THIS QUEUE
Visual Studio 2017	Hosted VS2017
Development tools on Ubuntu	Hosted Linux
Docker containers	Hosted Linux or Hosted VS2017
.NET Core	Hosted Linux (optimal) or Hosted VS2017
Development tools on macOS	Hosted macOS (see notes below)
Visual Studio 2013 or Visual Studio 2015	Hosted

Then, while editing your build pipeline, on the **Options** or **General** tab or **Process** step, for the **Agent queue**, select the queue you decided on.

Notes on choosing **Hosted macOS**:

- This option affects where your data is stored. Learn more
- For manual selection of tool versions on this Microsoft-hosted agent, see **Q & A** below.
- To disable the Hosted macOS agent pool for all projects, disable the `Hosted Agent` checkbox under Admin settings > Agent pools > Hosted macOS.
- To disable the Hosted macOS agent pool for a specific project, disable the `Hosted Agent` checkbox under Project settings > Agent pools > Hosted macOS.

Software

Software on Microsoft-hosted agents is updated once each 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 agent.
- Inventory of software currently installed on the Microsoft-hosted agent.

Capabilities and limitations

Microsoft-hosted agents:

- Have the above software. You can also add software using tool installers.
- Provide at least 10 GB of storage.
- Can run jobs for up to 6 hours (30 minutes on the free tier).
- Currently utilizing Microsoft Azure general purpose virtual machine sizes (Standard_DS2_v2 and Standard_DS3_v2)

Microsoft-hosted agents do not offer:

- The ability to log on.
- The ability to drop artifacts to a UNC file share.
- The ability to run XAML builds.
- Potential performance advantages that you might get by using self-hosted agents which might start and process builds faster. Learn more

If Microsoft-hosted agents don't meet your needs, then you can deploy your own self-hosted agents.

Avoid hard-coded references

When you use a Microsoft-hosted agent, 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 Microsoft-hosted agents (for example, the drive letter or folder that contains the repository).

Q & A

I can't select a Microsoft-hosted agent and I can't queue my build or deployment. How do I fix this?

By default, all project contributors in an organization have access to the Microsoft-hosted agents. But, your organization administrator may limit the access of Microsoft-hosted agents to select users or projects. Ask the owner of your VSTS organization to grant you permission to use a Microsoft-hosted agent. See agent queue security.

I need more agents. What can I do?

A: All VSTS organizations 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 jobs.

I'm looking for the Microsoft-hosted XAML build controller. Where did it go?

The Microsoft-hosted XAML build controller is no longer supported. If you have an organization in which you still need to run XAML builds, you should set up a self-hosted build server and a self-hosted build controller.

How can I manually select versions of tools on the Hosted macOS agent?

- **Xamarin**

  To manually select a Xamarin SDK version to use on the **Hosted macOS** agent, before your Xamarin build task, execute this command line as part of your build, replacing the Mono version number 5.4.1

as needed (also replacing '.' characters with underscores: '_'). Choose the Mono version that is associated with the Xamarin SDK version that you need.

```
/bin/bash -c "sudo $AGENT_HOMEDIRECTORY/scripts/select-xamarin-sdk.sh 5_4_1"
```

Mono versions associated with Xamarin SDK versions on the **Hosted macOS** agent can be found here.

Note that this command does not select the Mono version beyond the Xamarin SDK. To manually select a Mono version, see instructions below.

- **Xcode**

  If you use the Xcode task included with 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** agent, before your `xcodebuild` build task, execute this command line as part of your build, replacing the Xcode version number 8.3.3 as needed:

  ```
  /bin/bash -c "sudo xcode-select -s /Applications/Xcode_8.3.3.app/Contents/Developer"
  ```

  Xcode versions on the **Hosted macOS** agent can be found here.

- **Mono**

  To manually select a Mono version to use on the **Hosted macOS** agent, before your Mono build task, execute this script in each phase 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.

Videos

Deploy an agent on Windows

7/19/2018 • 7 minutes 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 Microsoft-hosted agent meets your needs, you can skip setting up a private Windows agent.
> - If your code is in an on-premises Team Foundation Server (TFS) 2015 server, see Deploy an agent on Windows for on-premises TFS 2015.
> - Otherwise, you've come to the right place to set up an agent on Windows. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see Build and release agents.

Check prerequisites

Make sure your machine is prepared with our Windows system prerequisites.

If you're building from a Subversion repo, you must install the Subversion client on the machine.

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 or newer

1. Sign in with the user account you plan to use in either your VSTS organization (`https://{your-organization}.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.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a deployment group agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

5. Copy the token. You'll use this token when you configure the agent.

Authenticate as a user of TFS 2017 or newer

You can use either a domain user or a local Windows user on each of your TFS application tiers.

Authenticate as a user of 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 organization 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 organization or TFS server:
   - VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?

2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator

can be an agent pool administrator, a VSTS organization owner, or a TFS server administrator. If it's a deployment group agent, the administrator can be an deployment group administrator, a VSTS organization owner, or a TFS server administrator. You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page 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_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?
3. Click **Download agent**.
4. On the **Get agent** dialog box, click **Windows**.
5. Click the **Download** button.
6. Follow the instructions on the page.

Server URL on VSTS

`https://{your-organization}.visualstudio.com`

Server URL on TFS 2017 and newer

`https://{your_server}/tfs`

Authentication type

VSTS

Choose **PAT**, and then paste the PAT token you created into the command prompt window.

TFS 2017 and newer

> **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. Use a personal access token (PAT) if your TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your TFS instance instead of the domain controller.

> **NOTE**
> When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent. Learn more at Communication with 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

What version of the agent runs with TFS 2017?

TFS VERSION	AGENT VERSION
2017 RTM	2.105.7
2017.3	2.112.0

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://{organization_name}.visualstudio.com
https://{organization_name}.vsrm.visualstudio.com
https://{organization_name}.pkgs.visualstudio.com
https://{organization_name}.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 set different environment variables for each individual agent?

Create a `.env` file under agent's root directory and put environment variables you want to set into the file as following format:

```
MyEnv0=MyEnvValue0
MyEnv1=MyEnvValue1
MyEnv2=MyEnvValue2
MyEnv3=MyEnvValue3
MyEnv4=MyEnvValue4
```

How do I configure the agent to bypass a web proxy and connect to 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 pipeline and scripts must still handle bypassing your web proxy for each task and tool you run in your build.
>
> For example, if you are using a NuGet task, you must configure your web proxy to support bypassing the URL for the server that hosts the NuGet feed you're using.

I'm using TFS and the URLs in the sections above don't work for me. Where can I get help?

Web site settings and security

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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

7/19/2018 • 6 minutes 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 organization. The parameter is used to specify the pool administrator credentials. The credentials are used to perform the pool administration changes and are not used later by the agent.

When using personal access tokens (PAT) authentication type, specify anything for the user name and specify the PAT as the password.

If passing the parameter from PowerShell, be sure to escape the semicolon or encapsulate the entire argument in quotes. For example: '/Login:user,password;AuthType=PAT'. Otherwise the semicolon will be interpreted by PowerShell to indicate the end of one statement and the beginning of another.

/NoPrompt
Indicates not to prompt and to accept the default for any values not provided on the command-line.

/WindowsServiceLogonAccount:WindowsServiceLogonAccount
Used for configuration commands to specify the identity to use for the Windows service. To specify a domain account, use the form Domain\SAMAccountName or the user principal name (for example user@fabrikam.com). Alternatively a built-in account can be provided, for example /WindowsServiceLogonAccount:"NT AUTHORITY\NETWORK SERVICE".

/WindowsServiceLogonPassword:WindowsServiceLogonPassword
Required if the /WindowsServiceLogonAccount parameter is provided.

/Configure

Configure supports the /NoPrompt switch for automated installation scenarios and will return a non-zero exit code on failure.

For troubleshooting configuration errors, detailed logs can be found in the _diag folder under the agent installation directory.

/ServerUrl:ServerUrl
The server URL should not contain the collection name. For example, `http://example:8080/tfs` or `https://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, vsiualstudio, 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

7/19/2018 • 10 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To build and deploy Xcode apps or Xamarin.iOS projects, you'll need at least one macOS agent. This agent can also build and deploy Java and Android apps.

Before you begin:

- If your build and release pipelines are in VSTS and a Microsoft-hosted agent meets your needs, you can skip setting up a private macOS agent.
- Otherwise, you've come to the right place to set up an agent on macOS. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see Build and release agents.

Check prerequisites

Make sure your machine is prepared with our macOS system prerequisites.

Prepare permissions

If you're building from a Subversion repo, you must install the Subversion client on the machine.

Decide which user you'll use

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 or newer

1. Sign in with the user account you plan to use in either your VSTS organization (`https://{your-organization}.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.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a deployment group agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

5. Copy the token. You'll use this token when you configure the agent.

Authenticate as a user of TFS 2017 or newer

You can use either a domain user or a local Windows user on each of your TFS application tiers.

Authenticate as a user of 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 organization 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 organization or TFS server:
   - VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?

2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator can

be an agent pool administrator, a VSTS organization owner, or a TFS server administrator. If it's a deployment group agent, the administrator can be an deployment group administrator, a VSTS organization owner, or a TFS server administrator. You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page 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_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?
3. Click **Download agent**.
4. On the **Get agent** dialog box, click **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-organization}.visualstudio.com`

TFS 2017 and newer: `https://{your_server}/tfs`

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.

> **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. Use a personal access token (PAT) if your TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your TFS instance instead of the domain controller.

> **NOTE**
> When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent on newer versions of TFS. Learn more at Communication with 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 organization. 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.{tfs-name}.{agent-name}
status vsts.agent.{tfs-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.agent.{tfs-name}.{agent-name}.plist

Started:
13472 0 vsts.agent.{tfs-name}.{agent-name}
```

The left number is the pid if the service is running. If second number is not zero, then a problem occurred.

Status

Command:

```
./svc.sh status
```

Output:

```
status vsts.agent.{tfs-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.{tfs-name}.{agent-name}.testsvc.plist

Started:
13472 0 vsts.agent.{tfs-name}.{agent-name}
```

The left number is the pid if the service is running. If second number is not zero, then a problem occurred.

Stop

Command:

```
./svc.sh stop
```

Output:

```
stopping vsts.agent.{tfs-name}.{agent-name}
status vsts.agent.{tfs-name}.{agent-name}:

/Users/{your-name}/Library/LaunchAgents/vsts.{tfs-name}.{agent-name}.testsvc.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 configure the service, it takes a snapshot of some useful environment variables for your current logon user such as PATH, LANG, JAVA_HOME, ANT_HOME, and MYSQL_PATH. If you need to update the variables (for example, after installing some new software):

1. ```
   ./env.sh
   ```

2. ```
   ./svc.sh stop
   ```

3. ```
   ./svc.sh start
   ```

The snapshot of the environment variables is stored in `.env` file under agent root directory, you can also change that file directly to apply environment variable changes.

Run instructions before the service starts

You can also run your own instructions and commands to run when the service starts. For example, you could set up the environment or call scripts.

1. Edit `runsvc.sh`.
2. Replace the following line with your instructions:

```
# insert anything to setup env when running as a service
```

Service Files
```

When you install the service, some service files are put in place.

.plist service file

A .plist service file is created:

`~/Library/LaunchAgents/vsts.agent.{tfs-name}.{agent-name}.plist`

For example:

`~/Library/LaunchAgents/vsts.agent.fabrikam.our-osx-agent.plist`

`sudo ./svc.sh install` generates this file from this template: `./bin/vsts.agent.plist.template`

.service file

`./svc.sh start` finds the service by reading the `.service` file, which contains the path to the plist service file described above.

Alternative service mechanisms

We provide the `./svc.sh` script as a convenient way for you to run and manage your agent as a launchd LaunchAgent service. But you can use whatever kind of service mechanism you prefer.

You can use the template described above as to facilitate generating other kinds of service files. For example, you modify the template to generate a service that runs as a launch daemon if you don't need UI tests and don't want to configure automatic log on and lock. See Apple Developer Library: Creating Launch Daemons and Agents.

Replace an agent

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://{organization_name}.visualstudio.com
https://{organization_name}.vsrm.visualstudio.com
https://{organization_name}.pkgs.visualstudio.com
https://{organization_name}.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 pipeline and scripts must still handle bypassing your web proxy for each task and tool you run in your build.
>
> For example, if you are using a NuGet task, you must configure your web proxy to support bypassing the URL for the server that hosts the NuGet feed you're using.

I'm using TFS and the URLs in the sections above don't work for me. Where can I get help?

Web site settings and security

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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

7/19/2018 • 10 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To build or deploy you'll need at least one agent. A Linux agent can build and deploy different kinds of apps, including Java and Android apps. We support Ubuntu, Red Hat, and CentOS.

> Before you begin:
>
> - If your build and release pipelines are in VSTS and a Microsoft-hosted agent meets your needs, you can skip setting up a private Linux agent.
> - Otherwise, you've come to the right place to set up an agent on Linux. Continue to the next section.

Learn about agents

If you already know what an agent is and how it works, feel free to jump right in to the following sections. But if you'd like some more background about what they do and how they work, see Build and release agents.

Check prerequisites

Where are your builds and releases running?

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 or newer

1. Sign in with the user account you plan to use in either your VSTS organization (`https://{your-organization}.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.

4. For the scope select **Agent Pools (read, manage)** and make sure all the other boxes are cleared. If it's a deployment group agent, for the scope select **Deployment group (read, manage)** and make sure all the other boxes are cleared.

5. Copy the token. You'll use this token when you configure the agent.

Authenticate as a user of TFS 2017 or newer

You can use either a domain user or a local Windows user on each of your TFS application tiers.

Authenticate as a user of 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 organization 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 organization or TFS server:
   - VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?

2. Click the pool on the left side of the page and then click **Roles**.
3. If the user account you're going to use is not shown, then get an administrator to add it. The administrator

can be an agent pool administrator, a VSTS organization owner, or a TFS server administrator. If it's a deployment group agent, the administrator can be an deployment group administrator, a VSTS organization owner, or a TFS server administrator. You can add a user to the deployment group administrator role in the **Security** tab on the **Deployment Groups** page 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_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?
3. Click **Download agent**.
4. On the **Get agent** dialog box, click **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-organization}.visualstudio.com`

TFS 2017 and newer: `https://{your_server}/tfs`

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.

> **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. Use a personal access token (PAT) if your TFS instance and the agent machine are not in a trusted domain. PAT authentication is handled by your TFS instance instead of the domain controller.

> **NOTE**
> When using PAT as the authentication method, the PAT token is used only for the initial configuration of the agent on newer versions of TFS. Learn more at Communication with 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 configure the service, it takes a snapshot of some useful environment variables for your current logon user such as PATH, LANG, JAVA_HOME, ANT_HOME, and MYSQL_PATH. If you need to update the variables (for example, after installing some new software):

1.
   ```
   ./env.sh
   ```
2.
   ```
   sudo ./svc.sh stop
   ```
3.
   ```
   sudo ./svc.sh start
   ```

The snapshot of the environment variables is stored in `.env` file under agent root directory, you can also change that file directly to apply environment variable changes.

Run instructions before the service starts

You can also run your own instructions and commands to run when the service starts. For example, you could set up the environment or call scripts.

1. Edit `runsvc.sh`.

2. Replace the following line with your instructions:

```
# insert anything to setup env when running as a service
```

Service files

When you install the service, some service files are put in place.

systemd service file

A systemd service file is created:

`/etc/systemd/system/vsts.agent.{tfs-name}.{agent-name}.service`

For example, you have configured an agent (see above) with the name `our-linux-agent`. The service file will be either:

- VSTS: the name of your organization. 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://{organization_name}.visualstudio.com
https://{organization_name}.vsrm.visualstudio.com
https://{organization_name}.pkgs.visualstudio.com
https://{organization_name}.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 pipeline and scripts must still handle bypassing your web proxy for each task and tool you run in your build.
>
> For example, if you are using a NuGet task, you must configure your web proxy to support bypassing the URL for the server that hosts the NuGet feed you're using.

I'm using TFS and the URLs in the sections above don't work for me. Where can I get help?

Web site settings and security

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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 self-hosted agent behind a web proxy

7/9/2018 • 3 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This topic explains how to run a v2 self-hosted agent behind a web proxy.

VSTS, TFS 2018 RTM and newer

(Applies to agent version 2.122 and newer.)

When your self-hosted agent is behind a web proxy, you can:

- Allow your agent to connect to 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 the `Get Source` task in builds and `Download Artifact` task in releases are also baked into the agent, those tasks will follow the agent proxy configuration from the `.proxy` file.

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 pipeline and scripts must still handle proxy configuration for each task and tool you run in your build. For example, if you are using a task that makes a REST API call, you must configure the proxy for that task.

Specify proxy bypass URLs

Create a `.proxybypass` file 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

7/3/2018 • 3 minutes to read • Edit Online

This topic explains how to run a v2 self-hosted agent with self-signed certificate.

Work with SSL server certificate

```
Enter server URL > https://corp.tfs.com/tfs
Enter authentication type (press enter for Integrated) >
Connecting to server ...
An error occurred while sending the request.
```

Agent diagnostic log shows:

```
[2017-11-06 20:55:33Z ERR  AgentServer] System.Net.Http.HttpRequestException: An error occurred while sending
the request. ---> System.Net.Http.WinHttpException: A security error occurred
```

This error may indicate the server certificate you used on your TFS server is not trusted by the build machine. Make sure you install your self-signed ssl server certificate into the OS certificate store.

```
Windows: Windows certificate store
Linux: OpenSSL certificate store
macOS: OpenSSL certificate store for agent version 2.124.0 or below
       Keychian 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 built with OpenSSL, More Detail

Git get sources fails with SSL certificate problem (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 approach to solve the problem.

1. Set the following git config in global level by the agent's run as user.

```
git config --global http."https://tfs.com/".sslCAInfo certificate.pem
```

> **NOTE**
> Setting system level git config is not reliable on Windows, 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 configure with 2.129.0 or higher version agent Pass `--gituseschannel` during agent configuration

```
./config.cmd --gituseschannel
```

> **NOTE**
> Git SChannel has more restrict requirment for your self-signed certificate. Self-singed certificate that generated by IIS or PowerShell command may not be capable with SChanel.

Work with SSL client certificate

IIS has a SSL setting that requires all incoming requests to TFS must present client certificate in addition to the regular credential.

When that IIS SSL setting 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 contains 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 contains the public key and signature of the Client certificate)
  - Client certificate private key in `.pem` format (This should contains only the private key of the Client certificate)
  - Client certificate archive package in `.pfx` format (This should contains 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 Keychian
- 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.

Artifacts in Team Build

7/9/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2015.3 and newer | TFS 2015 RTM (see Q&A)

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 pipeline.

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.

> 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.SourcesDirectory)
```

- 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.

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 RTM?

If you're using TFS 2015 RTM, then the steps in the above examples are not available. Instead, you copy and publish your artifacts using a single task: Build: Publish Build Artifacts.

Build pipeline history

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

From the **History** tab you can see a list of changes that includes who made the change and when the change occurred.

TFS 2017.3 and newer

To work with a change, select it, click **...** , and then click **Compare Difference** or **Revert Pipeline**.

TFS 2017 RTM

After you've viewed the history, if you want details about a change, select it and then choose **Diff**. If you want to roll back to an earlier version, select it, and then click **Rollback**.

Q & A

Can I edit the JSON source directly?

No

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 pipeline options

7/17/2018 • 6 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Here we explain settings you can change on the build pipeline **Options** tab.

Description

> **Team Foundation Server (TFS) 2017.1 and older**
>
> This section is available under **General tab.**

If you specify a description here, it is shown near the name of the build pipeline when you select it in the Build area of your project.

Build number format

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:

- Project name: Fabrikam
- Build pipeline name: CIBuild
- Branch: master
- Build ID: 752
- Date: August 5, 2009.
- Time: 9:07:03 PM.
- The build ran once earlier today.

If you specify this build number format:

```
$(TeamProject)_$(BuildDefinitionName)_$(SourceBranchName)_$(Date:yyyyMMdd)$(Rev:.r)
```

Then the second completed build on this day would be named: **Fabrikam_CIBuild_master_20090805.2**

Tokens

The following table shows how each token is resolved based on the previous example.

TOKEN	EXAMPLE REPLACEMENT VALUE
`$(BuildDefinitionName)`	CIBuild Note: The build pipeline name must not contain invalid or whitespace characters.
`$(BuildID)`	752 $(BuildID) is an internal immutable ID.
`$(DayOfMonth)`	5
`$(DayOfYear)`	217
`$(Hours)`	21
`$(Minutes)`	7
`$(Month)`	8
`$(Rev:.r)`	2 (The third build on this day will be 3, and so on.) Use **$(Rev:.r)** to ensure that every completed build has a unique name. When a build is completed, if nothing else in the build number has changed, the Rev integer value is incremented by one. If you want to show prefix zeros in the number, you can add additional **'r'** characters. For example, specify **$(rev:.rr)** if you want the Rev number to begin with 01, 02, and so on.
`$(Date:yyyyMMdd)`	20090824 You can specify other date formats such as **$(Date:MMddyy)**
`$(Seconds)`	3
`$(SourceBranchName)`	master
`$(TeamProject)`	Fabrikam
`$(Year:yy)`	09
`$(Year:yyyy)`	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 pipeline.
3. When the **Show url** link appears, click it and copy the URL to your clipboard.
4. Use the URL as the source of an image in the HTML of the page of the external web site. For example: `<img src="{URL you copied from the link}"/>`

Create a work item on failure

If the build process fails, you can automatically create a work item to track getting the problem fixed. You can specify the work item type.

You can also select if you want to assign the work item to the requestor. For example, if this is a CI build, and a team member checks in some code that breaks the build, then the work item is assigned to that person.

Additional Fields: You can set the value of work item fields. For example:

FIELD	VALUE
`System.Title`	`Build $(Build.BuildNumber) failed`
`System.Reason`	`Build failure`

Q: What other work item fields can I set? **A:** Work item field index

Allow scripts to access the OAuth token

Select this check box if you want to enable your script to use the build process OAuth token.

For an example, see Use a script to customize your build process.

Default agent queue

> **TFS 2017.1 and older**
>
> This section is available under **General tab.**

Select the queue that's attached to the pool that contains the agents you want to run this pipeline.

Tip: If your code is in Visual Studio Team Services (VSTS) and you run your builds on Windows, in many cases the simplest option is to use the Hosted pool.

Build job authorization scope

> **TFS 2017.1 and older**

> This section is available under **General tab.**

Specify the authorization scope for a build job. Select:

- **Project Collection** if the build needs access to multiple projects.
- **Current Project** if you want to restrict this build to have access only the resources in the current project.

Build job timeout in minutes

> **TFS 2017.1 and older**
>
> This section is available under **General tab.**

Specify the maximum time a build job is allowed to execute on an agent before being canceled by the server. Leave it empty or at zero if you want the job to never be canceled by the server.

Build job timeout in minutes (VSTS, TFS 2017.3 and newer)

Specify the maximum time a build job is allowed to respond after the a user cancels the build. You can specify a value from 1 to 60 minutes.

Set this value to allow sufficient time for tasks to complete in cases where you've specified to **Run this task** as **Even if a previous task has failed, even if the build was cancelled** or as **Custom conditions** that allow a task to always run after cancellation.

Whatever value you set here, the **Build job timeout in minutes** limit still applies. For example:

- Build job timeout in minutes = 30
- Build job cancel timeout in minutes = 15
- A user clicks **Cancel** after the build has run for 25 minutes.
- The build is given 5 minutes to cancel instead of the 15 you specified.

> **NOTE**
>
> The system typically consumes about 10 seconds of this time allotment for messaging before your tasks run.

Demands

> **TFS 2017.1 and older**
>
> This section is available under **General tab.**

Use demands to make sure that the capabilities your build needs are present on the build agents that run it. Demands are asserted automatically by build tasks or manually by you.

Build task demands

Some build tasks won't run unless one or more demands are met by the build agent. For example, the Visual Studio Build task demands that `msbuild` and `visualstudio` are installed on the build agent. If your build includes tasks that have demands, they are listed first.

Manually entered demands

You might need to use on-premises build agents with special capabilities. For example, your build process requires SpecialSoftware.

Add the demand to your build pipeline.

NAME	TYPE
SpecialSoftware	exists

Register each build agent that has the capability.

1. Go to the Agent pools tab at the root of the control panel:
   - VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?
2. Click the agent, and then click **Capabilities**
3. Add something like the following entry:

FIRST BOX	SECOND BOX
SpecialSoftware	C:\Program Files (x86)\SpecialSoftware

> **Tip:** When you manually queue a build you can change the demands for that run.

Multi-configuration

Select this option to build multiple configurations. For example, you could build a C++ app for both debug and release configurations on both x86 and x64 platforms. In VSTS and TFS 2018, you set this option in the **Tasks** tab for each **Phase** in your pipeline (not in the **Options** tab). To learn about multi-configuration, see the example Build your C++ app for Windows.

Q & A

In what time zone are the build number time values expressed?

If you are using VSTS, then the time zone is UTC.

If you are using an on-premises Team Foundation Server, then the time zone is the same as the time zone of the operating system of the machine where you are running your application tier server.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 pipeline source repositories

7/19/2018 • 16 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

At the beginning of the build process, the agent downloads files from your remote repository into a local sources directory.

If your build consists of multiple jobs (for example, because of multiple phases), the agent downloads the files at the beginning of each job. You can specify only a single source repository for your entire build process.

VSTS and TFS 2018: To specify the source repository, click the **Tasks** tab, and then click **Get sources**, and then select the type of repo that contains your source files.

TFS 2017: To specify the source repository:

- **TFS 2017.3** Click the **Tasks** tab, and then click **Get sources**, and then select the type of repo that contains your source files.
- **TFS 2017 RTM** Click the **Repository** tab, and then select the type of repo that contains your source files.

TFS 2015: To specify the source repository, click the **Repository** tab, and then select the type of repo that contains your source files.

You can choose from the following repository types:

REPOSITORY TYPE	VSTS (YAML)	VSTS	TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Git repo in a project	Yes	Yes	Yes	Yes
Git repo in Bitbucket Cloud	No	Yes	No	No
Git repo in GitHub	Yes	Yes	No	No
Git repo in GitHub Enterprise	No	Yes	No	No
Git repo (external/remote)	No	Yes	Yes	Yes
Subversion	No	Yes	Yes	No

REPOSITORY TYPE	VSTS (YAML)	VSTS	TFS 2018, TFS 2017, TFS 2015.4	TFS 2015 RTM
Team Foundation Version Control (TFVC) repo in a project	No	Yes	Yes	Yes

> **NOTE**
> To build code from Subversion, you must install a Subversion client (`svn`) on your build agents.

Git options

When you select a Git repo (in a project, GitHub, Bitbucket, or remote Git repo), you've got the following options.

FEATURE	VSTS	TFS 2018	TFS 2017.2	TFS 2017 RTM	TFS 2015.4	TFS 2015 RTM
Branch	Yes	Yes	Yes	Yes	Yes	Yes
Clean	Yes	Yes	Yes	Yes	Yes	Yes
Tag or label sources	Project	Team project	Team project	Team project	Team project	No
Report build status	Yes	Yes	Yes	Yes	No	No
Checkout submodules	Yes	Yes	Yes	Yes	Yes	Yes
Checkout files from LFS	Yes	Yes	Yes	macOS and Linux agents	macOS and Linux agents	macOS and Linux agents
Don't sync sources	Yes	Yes	Yes	No	No	No
Shallow fetch	Yes	Yes	Yes	macOS and Linux agents	macOS and Linux agents	macOS and Linux agents

> **NOTE**
> **VSTS, TFS 2017.2 and newer:** Click **Advanced settings** in the **Get Sources** task to see some of the above options.
> **VSTS (YAML):** Not all of these options are available yet. For available options and syntax, refer to YAML documentation.

Branch

TFS 2017 RTM and TFS 2015: This field is called **Default branch**.

This is the branch that you want to be the default when you manually queue this build. If you set a scheduled trigger for the build, this is the branch from which your build will get the latest sources. The default branch has no bearing when the build is triggered through continuous integration (CI). Usually you'll set this to be the same as the default branch of the repository (for example, "master").

Clean the local repo on the agent

You can perform different forms of cleaning the working directory of your self-hosted agent before a build runs.

In general, for faster performance of your self-hosted agents, don't clean the repo. In this case, to get the best performance, make sure you're also building incrementally. For example, if you're 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), your options are below.

> **NOTE**
>
> Cleaning is not effective if you're using a Microsoft-hosted agent because you'll get a new agent every time.

VSTS, TFS 2018, TFS 2017.2, TFS 2017.3

Select one of the following options:

- **Sources**: The build process performs an undo of any changes in `$(Build.SourcesDirectory)`. More specifically, the following Git commands are executed prior to fetching the source.

  ```
  git clean -fdx
  git reset --hard HEAD
  ```

- **Sources and output directory**: Same operation as **Sources** option above, plus: Deletes and recreates `$(Build.BinariesDirectory)`. Note that the `$(Build.ArtifactStagingDirectory)` and `$(Common.TestResultsDirectory)` are always deleted and recreated prior to every build regardless of any of these settings.
- **Sources directory**: Deletes and recreates `$(Build.SourcesDirectory)`. This results in initializing a new, local Git repository for every build.
- **All build directories**: Deletes and recreates `$(Agent.BuildDirectory)`. This results in initializing a new, local Git repository for every build.

TFS 2017 RTM

If you select **True** then the build process performs an undo of any changes. If errors occur, then it deletes the contents of `$(Build.SourcesDirectory)`.

If you want the Clean switch described above to work differently, then on the **Variables** tab, define the `Build.Clean` variable and set its value to:

- `all` if you want to delete `$(Agent.BuildDirectory)`, which is the entire working folder that contains the sources folder, binaries folder, artifact folder, and so on.
- `source` if you want to delete `$(Build.SourcesDirectory)`.
- `binary` If you want to delete `$(Build.BinariesDirectory)`.

TFS 2015.4

If you select **True** then the build process performs an undo of any changes. If errors occur, then it deletes the contents of `$(Build.SourcesDirectory)`.

If you want the Clean switch described above to work differently, then on the **Variables** tab, define the `Build.Clean` variable and set its value to:

- `all` if you want to delete `$(Agent.BuildDirectory)`, which is the entire working folder that contains the

sources folder, binaries folder, 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.

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)`.

Label sources

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

> **NOTE**
> You can only use this feature when the source repository in your build is a Git or TFVC repository from your project.

In the **Label format** you can use user-defined and predefined variables that have a scope of "All." For example:

```
$(Build.DefinitionName)_$(Build.DefinitionVersion)_$(Build.BuildId)_$(Build.BuildNumber)_$(My.Variable)
```

The first four variables are predefined. `My.Variable` is defined by you on the variables tab.

The build process labels your sources with a Git tag.

Some build variables might yield a value that is not a valid label. For example, variables such as `$(Build.RequestedFor)` and `$(Build.DefinitionName)` can contain white space. If the value contains white space, the tag is not created.

After the sources are tagged by your build process, an artifact with the Git ref `refs/tags/{tag}` is automatically added to the completed build. This gives your team additional traceability and a more user-friendly way to navigate from the build to the code that was built.

Report build status (VSTS, TFS 2017 and newer)

You've got the option to give your team a view of the build status from your remote source repository.

If your sources are in a Git repository in your project, then this option displays a badge in the **Code** hub to indicate whether the build is passing or failing. The build status is displayed in the following tabs:

- **Files**: Indicates the status of the latest build for the selected branch.
- **Commits**: Indicates the build status of the each commit (this requires continuous integration (CI) trigger to be enabled for your builds).
- **Branches**: Indicates the status of the latest build for each branch.

If you use multiple build pipelines for the same repository in your project, then you may choose to enable this option for one or more of the pipelines. In the case when this option is enabled on multiple pipelines, the badge

in the **Code** hub indicates the status of the latest build across all the pipelines. Your team members can click the build status badge to view the latest build status for each one of the build pipelines.

GitHub (VSTS)

If your sources are in GitHub, then this option publishes the status of your build to GitHub using GitHub status APIs. If your build is triggered from a GitHub pull request, then you can view the status on the GitHub pull request page. This also allows you to set status policies within GitHub and automate merges. If your build is triggered by continuous integration (CI), then you can view the build status on the commit or branch in GitHub.

Other types of Git remote repositories

If your source is in any other type of remote repository, then you cannot use VSTS or TFS to automatically publish the build status. However, you can use a build badge as a way to integrate and show build status within your version control experiences.

Checkout submodules

Select if you want to download files from submodules. You can either choose to get the immediate submodules or all submodules nested to any depth of recursion.

The build process will check out your Git submodules as long as they are:

- **Unauthenticated:** A public, unauthenticated repo with no credentials required to clone or fetch.
- **Authenticated:**
  - Contained in the same project, GitHub organization, or Bitbucket account as the Git repo specified above.
  - Added by using a URL relative to the main repository. For example, this one would be checked out: `git submodule add /../../submodule.git mymodule` This one would not be checked out:
    ```
    git submodule add https://fabrikamfiber.visualstudio.com/DefaultCollection/_git/ConsoleApp
    mymodule
    ```

> **NOTE**
>
> If you're running **TFS 2017.1, TFS 2017 RTM, or TFS 2015**, then the submodules must be children (immediate submodules)** of the Git repo you've selected for this build process. In effect, the build process runs `git submodule update --init` (not `git submodule update -init --recursive`).

Authenticated submodules

> **NOTE**
>
> Make sure that you have registered your submodules using HTTPS and not using SSH.

The same credentials that are used by the agent to get the sources from the main repository are also used to get the sources for submodules.

If your main repository and submodules are in a Git repository in your VSTS project, then you can select the account used to access the sources. On the **Options** tab, on the **Build job authorization scope** menu, select either:

- **Project collection** to use the Project Collection Build service account
- **Current project** to use the Project Build Service account.

Make sure that whichever account you use has access to both the main repository as well as the submodules.

If your main repository and submodules are in the same GitHub organization, then the token stored in the

GitHub service connection is used to access the sources.

Alternative to using the Checkout submodules option

In some cases you can't use the Checkout submodules option. You might have a scenario where a different set of credentials are needed to access the submodules. This can happen, for example, if your main repository is in VSTS and your submodules are in GitHub, if your main repository is in GitHub and your submodules are in VSTS, or if your submodules are in a different VSTS organization than your main repository.

If you can't use the Checkout submodules option, then you can instead use a custom script with the following Git command to get the sources for submodules onto your agent.

```
git -c http.https://<url of submodule repository>.extraheader="AUTHORIZATION: basic ********" submodule
update --init --recursive
```

Use a secret variable in your project or build pipeline 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 isn't desirable during automated builds when user interaction isn't possible.

Checkout files from LFS

Select if you want to download files from large file storage (LFS).

- **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 self-hosted 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 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 project is mapped in the workspace. This configuration results in the build agent downloading all the files in the version control folder of your project. If this folder contains lots of data, your build could waste build system resources and slow down your build 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 forms of cleaning the working directory of your self-hosted agent before a build

runs.

In general, for faster performance of your self-hosted agents, don't clean the repo. In this case, to get the best performance, make sure you're also building incrementally. For example, if you're 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), your options are below.

> **NOTE**
> Cleaning is not relevant if you are using a Microsoft-hosted agent because you get a new agent every time in that case.

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 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?

Scorch 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.

Build pipeline triggers

7/19/2018 • 6 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

On the **Triggers** tab you specify the events that will trigger the build. You can use the same build pipeline for both CI and scheduled builds.

Continuous integration (CI)

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.

> You can batch changes when your code is in Git in the 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. If you want to use wildcard characters, then type the branch specification (for example, `features/modules/*`) and then press Enter.

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, TFS 2017.3 and newer

TFS 2017.1 and older versions

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. If you want to use wildcard characters, then type the branch specification (for example, `features/modules/*`) and then press Enter.

Example: Nightly build of Git repo in multiple time zones

VSTS, TFS 2017.3 and newer versions

TFS 2017.1 and older versions

Example: Nightly build with different frequencies

VSTS, TFS 2017.3 and newer versions

TFS 2017.1 and older versions

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 developers try to check-in, they are prompted to build their changes.

The system then creates a shelveset and builds it.

For details on the gated check-in experience, see Check in to a folder that is controlled by a gated check-in build 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 Microsoft-hosted agent or a self-hosted agent.

Build completion triggers

Large products have several components that are dependent on each other. These components are often independently built. When an upstream component (a library, for example) changes, the downstream dependencies have to be rebuilt and revalidated.

In situations like these, add a build completion trigger to run your build upon the successful completion of the **triggering build**. You can select any other build in the same project.

After you add a **build completion** trigger, select the **triggering build**. If the triggering build is sourced from a Git repo, you can also specify **branch filters**. If you want to use wildcard characters, then type the branch specification (for example, `features/modules/*`) and then press Enter.

> **NOTE**
> Keep in mind that in some cases, a single multi-phase build could meet your needs. However, a build completion trigger is useful if your requirements include different configuration settings, options, or a different team to own the dependent process.

Download artifacts from the triggering build

In many cases you'll want to download artifacts from the triggering build. To do this:

1. Edit your build pipeline.
2. Add the **Download Build Artifacts** task to one of your phases under **Tasks**.
3. For **Download artifacts produced by**, select **Specific build**.
4. Select the team **Project** that contains the triggering build pipeline.
5. Select the triggerging **Build pipeline**.
6. Select **When appropriate, download artifacts from the triggering build**.
7. Even though you specified that you want to download artifacts from the triggering build, you must still select a value for **Build**. The option you choose here determines which build will be the source of the artifacts whenever your triggered build is run because of any other reason than `BuildCompletion` (e.g. `Manual`, `IndividualCI`, or `Schedule`, and so on).
8. Specify the **Artifact name** and make sure it matches the name of the artifact published by the triggering build.
9. Specify the **Destination directory** to which you want to download the artifacts. For example: `$(Build.BinariesDirectory)`

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?

Someone must view a page in your VSTS organization regularly for CI and scheduled builds to run. It can be any page, including, for example, the **Build and release** hub.

Your VSTS organization goes dormant five minutes after the last user signed out. After that, each of your build pipelines will run one more time. For example, while your organization is dormant:

- A nightly build of code in your VSTS organization will run only one night until someone signs in again.
- CI builds of an external Git repo will stop running until someone signs in again.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/17/2018 • 15 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

> **NOTE**
> Some features and predefined variables are not available in certain versions of TFS. We're working on updating this topic to call out these differences.

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 tasks	Yes	Yes	Yes	`$(Build.DefinitionName)`	Command line, Copy files
Apply a version control label during the build process	Yes	Yes	No	`$(Build.DefinitionName)`	Repository tab (Git and Team Foundation version control **Label format**)
Customize the build number	Yes	Yes	No	`$(Build.DefinitionName)`	Build number format options
Environment variable in Windows batch scripts	Yes	Yes	Yes	`%BUILD_DEFINITIONNAME%`	Batch script
Environment variable in PowerShell scripts	Yes	Yes	Yes	`$env:BUILD_DEFINITIONNAME`	PowerShell script
Environment variable in Shell scripts	Yes	Yes	Yes	`$BUILD_DEFINITIONNAME`	Shell script

User-defined variables

Variables are a great way to store and share key bits of data in your build pipeline. 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.

TFS 2017.2, TFS 2017.3

TFS 2017 RTM

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 tasks are not given access by default.
- Decrypted for access by your build tasks. So you can use them in password arguments and also pass them explicitly into a script or a program from your build task (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. Note that the updated variable value is scoped to the phase being executed, and does not flow across phases or environments.

> **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

Build.Clean

Modifies how the build agent cleans things up. See Source repositories.

System.Debug

If you need more detailed logs to debug build problems, define and set it to `true`.

Environment variables

You can pass environment variables of the build machine into build tasks. For example, on the Build tab of a build pipeline, add this task:

TASK	ARGUMENTS
Utility: Command Line	Tool: `echo` Arguments: `$(PATH)`

> **NOTE**
> If you have defined 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

Agent.BuildDirectory

Environment variable: AGENT_BUILDDIRECTORY

Scope: Agent

The local path on the agent where all folders for a given build pipeline are created.

For example: `c:\agent\_work\1`

For example:

- In TFS 2015.4: `C:\TfsData\Agents\Agent-MACHINENAME\_work\1`
- In TFS 2015 RTM: `C:\TfsData\Build\_work\6c3842c6`

Agent.HomeDirectory

Environment variable: AGENT_HOMEDIRECTORY

Scope: Agent

The directory the agent is installed into. This contains the agent software.

For example: `c:\agent`

For example:

- In TFS 2015.4: `C:\TfsData\Agents\Agent-MACHINENAME`
- In TFS 2015 RTM:
  - Pre-configured agent: `C:\Program Files\Microsoft Team Foundation Server 14.0\Build`
  - Agent installed by you: `C:\Agent`

This directory is specified by you. See Agents.

Agent.Id

Environment variable: AGENT_ID

Scope: Agent

The ID of the agent.

Agent.JobStatus

Environment variable: AGENT_JOBSTATUS or agent.jobstatus

Environment variable: agent.jobstatus

Scope: Agent

The status of the build.

- `Canceled`
- `Failed`
- `Succeeded`
- `SucceededWithIssues` (partially successful)

Agent.MachineName

Environment variable: AGENT_MACHINENAME

Scope: Agent

The name of the machine on which the agent is installed.

> **NOTE**
>
> This variable is available in **TFS 2015.4**, not in **TFS 2015 RTM**.

Agent.Name

Environment variable: AGENT_NAME

Scope: 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 agents(../concepts/agents/agents.md).

Agent.WorkFolder

Environment variable: AGENT_WORKFOLDER

Scope: Agent

The working directory for this agent. For example: `c:\agent\_work`.

Build.ArtifactStagingDirectory

Environment variable: BUILD_ARTIFACTSTAGINGDIRECTORY

Scope: Agent

The local path on the agent where any artifacts are copied to before being pushed to their destination. For example:

- TFS 2015.4: `C:\TfsData\Agents\Agent-MACHINENAME\_work\1\a`
- TFS 2015 RTM default agent: C:\TfsData\Build_work\6c3842c6\artifacts
- TFS 2015 RTM agent installed by you: C:\Agent_work\6c3842c6\artifacts

This directory is purged before each new build, so you don't have to clean it up yourself.

A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks. See Artifacts in Team Build.

In **TFS 2015.4**, Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable.

The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: `c:\agent\_work\1\a`

A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.

> **NOTE**
>
> Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.
>
> See Artifacts in Team Build.

Build.BuildId

Environment variable: BUILD_BUILDID

Scope: All

The ID of the record for the completed build.

Build.BuildNumber

Environment variable: BUILD_BUILDNUMBER

Scope: Agent, label format (see Notes)

The name of the completed build. You can specify the build number format that generates this value in the pipeline options.

A typical use of this variable is to make it part of the label format, which you specify on the repository tab.

Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.

Build.BuildUri

Environment variable: BUILD_BUILDURI

Scope: Agent

The URI for the build. For example: `vstfs:///Build/Build/1430`.

Build.BinariesDirectory

Build.BinariesDirectory (TFS 2015.4)

Environment variable: BUILD_BINARIESDIRECTORY

Scope: Agent

The local path on the agent you can use as an output folder for compiled binaries.

For example: `c:\agent\_work\1\b`.

For example: `C:\TfsData\Agents\Agent-MACHINENAME\_work\1\b`

By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up

on the Repository tab.

Build.DefinitionName

Environment variable: BUILD_DEFINITIONNAME

Scope: All (see Notes)

The name of the build pipeline.

Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.

Build.DefinitionVersion

Environment variable: BUILD_DEFINITIONVERSION

Scope: All

The version of the build pipeline.

Build.QueuedBy

Environment variable: BUILD_QUEUEDBY

Scope: All (see Notes)

How are the identity variables set?

Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.

Build.QueuedById

Environment variable: BUILD_QUEUEDBYID

Scope: All

How are the identity variables set?

Build.Reason

Environment variable: BUILD_REASON

Scope: All

> **VSTS Only**

The event that caused the build to run.

- `Manual`: A user manually queued the build.
- `IndividualCI`: **Continuous integration (CI)** triggered by a Git push or a TFVC check-in.
- `BatchedCI`: **Continuous integration (CI)** triggered by a Git push or a TFVC check-in, and the **Batch changes** was selected.
- `Schedule`: **Scheduled** trigger.
- `ValidateShelveset`: A user manually queued the build of a specific TFVC shelveset.
- `CheckInShelveset`: **Gated check-in** trigger.
- `PullRequest`: The build was triggered by a Git branch policy that requires a build. ::: moniker range="vsts"
- `BuildCompletion`: The build was triggered by another build ::: moniker-end

See Build pipeline triggers, Improve code quality with branch policies.

Build.Repository.Clean

Environment variable: BUILD_REPOSITORY_CLEAN

Scope: Agent

The value you've selected for **Clean** in the source repository settings.

Build.Repository.LocalPath

Environment variable: BUILD_REPOSITORY_LOCALPATH

Scope: Agent

The local path on the agent where your source code files are downloaded. For example: `c:\agent\_work\1\s`

By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.

Build.Repository.Name

Environment variable: BUILD_REPOSITORY_NAME

Scope: Agent

The name of the repository.

Build.Repository.Provider

Environment variable: BUILD_REPOSITORY_PROVIDER

Scope: Agent

The type of repository you selected.

- `TfsGit`: TFS Git repository
- `TfsVersionControl`: Team Foundation Version Control
- `Git`: Git repository hosted on an external server
- `GitHub`
- `Svn`: Subversion

Build.Repository.Tfvc.Workspace

Environment variable: BUILD_REPOSITORY_TFVC_WORKSPACE

Scope: Agent

Defined if your repository is Team Foundation Version Control. The name of the TFVC workspace used by the build agent.

For example, if the Agent.BuildDirectory is `c:\agent\_work\12` and the Agent.Id is `8`, the workspace name could be: `ws_12_8`

Build.Repository.Uri

Environment variable: BUILD_REPOSITORY_URI

Scope: Agent

The URL for the repository. For example:

- Git: `https://fabrikamfiber.visualstudio.com/_git/Scripts`
- TFVC: `https://fabrikamfiber.visualstudio.com/`

Build.RequestedFor

Environment variable: BUILD_REQUESTEDFOR

Scope: All (see Notes)

How are the identity variables set?

Note: This value can contain whitespace or other invalid label characters. In these cases, the label format will fail.

Build.RequestedForEmail

Environment variable: BUILD_REQUESTEDFOREMAIL

Scope: All

How are the identity variables set?

Build.RequestedForId

Environment variable: BUILD_REQUESTEDFORID

Scope: All

How are the identity variables set?

Build.SourceBranch

Environment variable: BUILD_SOURCEBRANCH

Scope: All (see Notes)

The branch the build was queued for. Some examples:

- Git repo branch: `refs/heads/master`
- Git repo pull request: `refs/pull/1/merge`
- TFVC repo branch: `$/teamproject/main`
- TFVC repo gated check-in: `Gated_2016-06-06_05.20.51.4369;username@live.com`
- TFVC repo shelveset build: `myshelveset;username@live.com`

When you use this variable in your build number format, the forward slash characters (`/`) are replaced with underscore characters `_`).

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.

Build.SourceBranchName

Environment variable: BUILD_SOURCEBRANCHNAME

Scope: All (see Notes)

The name of the branch the build was queued for.

- Git repo branch or pull request: The last path segment in the ref. For example, in `refs/heads/master` this value is `master`.
- TFVC repo branch: The last path segment in the root server path for the workspace. For example in `$/teamproject/main` this value is `main`.
- TFVC repo gated check-in or shelveset build is the name of the shelveset. For example, `Gated_2016-06-06_05.20.51.4369;username@live.com` or `myshelveset;username@live.com`.

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.

Build.SourcesDirectory

Environment variable: BUILD_SOURCESDIRECTORY

Scope: Agent

The local path on the agent where your source code files are downloaded. For example: `c:\agent\_work\1\s`

By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.

Build.SourceVersion

Environment variable: BUILD_SOURCEVERSION

Scope: Agent

The latest version control change that is included in this build.

- Git: The commit ID.
- TFVC: the changeset.

Build.StagingDirectory

Environment variable: BUILD_STAGINGDIRECTORY

Scope: Agent

TFS 2015 RTM

The local path on the agent you can use as an output folder for compiled binaries. For example: `C:\TfsData\Build\_work\6c3842c6\staging`.

By default, new build pipelines are not set up to clean this directory. You can define your build to clean it up on the Repository tab.

TFS 2015.4

The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: `C:\TfsData\Agents\Agent-MACHINENAME\_work\1\a`

This directory is purged before each new build, so you don't have to clean it up yourself.

A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks. See Artifacts in Team Build.

In **TFS 2015.4**, Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable.

The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: `c:\agent\_work\1\a`

A typical way to use this folder is to publish your build artifacts with the Copy files and Publish build artifacts tasks.

> **NOTE**
>
> Build.ArtifactStagingDirectory and Build.StagingDirectory are interchangeable. This directory is purged before each new build, so you don't have to clean it up yourself.
>
> See Artifacts in Team Build.

Build.Repository.Git.SubmoduleCheckout

Environment variable: BUILD_REPOSITORY_GIT_SUBMODULECHECKOUT

Scope: Agent

The value you've selected for **Checkout submodules** on the repository tab.

Build.SourceTfvcShelveset

Environment variable: BUILD_SOURCETFVCSHELVESET

Scope: All (see Notes)

Defined if your repository is Team Foundation Version Control.

If you are running a gated build or a shelveset build, this is set to the name of the shelveset you are building.

Note: This variable yields a value that is invalid for build use in a build number format

Common.TestResultsDirectory

Environment variable: COMMON_TESTRESULTSDIRECTORY

Scope: Agent

The local path on the agent where the test results are created. For example: `c:\agent\_work\1\TestResults`

Build.TriggeredBy.BuildId

Environment variable: BUILD_TRIGGEREDBY_BUILDID

Scope: Agent

If the build was triggered by another build, then this variable is set to the BuildID of the triggering build.

Build.TriggeredBy.DefinitionId

Environment variable: BUILD_TRIGGEREDBY_DEFINITIONID

If the build was triggered by another build, then this variable is set to the DefinitionID of the triggering build.

Build.TriggeredBy.BuildDefinitionName

Environment variable: BUILD_TRIGGEREDBY_DEFINITIONNAME

If the build was triggered by another build, then this variable is set to the name of the triggering build pipeline.

Build.TriggeredBy.BuildNumber

Environment variable: BUILD_TRIGGEREDBY_BUILDNUMBER

If the build was triggered by another build, then this variable is set to the number of the triggering build.

Build.TriggeredBy.ProjectID

Environment variable: BUILD_TRIGGEREDBY_PROJECTID

If the build was triggered by another build, then this variable is set to ID of the project that contains the triggering build.

System.AccessToken

Environment variable: SYSTEM_ACCESSTOKEN

Scope: Agent

Use the OAuth token to access the REST API.

System.CollectionId

Environment variable: SYSTEM_COLLECTIONID

Scope: All

The GUID of the team foundation collection.

System.DefaultWorkingDirectory

Environment variable: SYSTEM_DEFAULTWORKINGDIRECTORY

Scope: Agent

The local path on the agent where your source code files are downloaded. For example: `c:\agent\_work\1\s`

By default, new build pipelines update only the changed files. You can modify how files are downloaded on the Repository tab.

System.DefinitionId

Environment variable: SYSTEM_DEFINITIONID

Scope: All

The ID of the build pipeline.

System.HostType

Environment variable: SYSTEM_HOSTTYPE

Scope: All

Set to `build` if the process is a build or `release` if the process is a release.

System.PullRequest.IsFork

Environment variable: SYSTEM_PULLREQUEST_ISFORK

Scope: All

If the pull request is from a fork of the repository, this variable is set to `True`. Otherwise, it is set to `False`.

System.PullRequest.PullRequestId

Environment variable: SYSTEM_PULLREQUEST_PULLREQUESTID

Scope: All

The ID of the pull request that caused this build. For example: `17`. (This variable is initialized only if the build ran because of a Git PR affected by a branch policy.)

System.PullRequest.PullRequestNumber

Environment variable: SYSTEM_PULLREQUEST_PULLREQUESTNUMBER

Scope: All

The number of the pull request that caused this build. This variable is populated for pull requests from GitHub which have a different pull request ID and pull request number.

System.PullRequest.SourceBranch

Environment variable: SYSTEM_PULLREQUEST_SOURCEBRANCH

Scope: All

The branch that is being reviewed in a pull request. For example: `refs/heads/users/raisa/new-feature`. (This variable is initialized only if the build ran because of a Git PR affected by a branch policy.)

System.PullRequest.SourceRepositoryURI

Environment variable: SYSTEM_PULLREQUEST_SOURCEREPOSITORYURI

Scope: All

> **VSTS Only**

The URL to the repo that contains the pull request. For example: `https://ouraccount.visualstudio.com/_git/OurProject`. (This variable is initialized only if the build ran because of a VSTS Git PR affected by a branch policy. It is not initialized for GitHub PRs.)

System.PullRequest.TargetBranch

Environment variable: SYSTEM_PULLREQUEST_TARGETBRANCH

Scope: All

The branch that is the target of a pull request. For example: `refs/heads/master`. (This variable is initialized only if the build ran because of a Git PR affected by a branch policy.)

System.TeamFoundationCollectionUri

Environment variable: SYSTEM_TEAMFOUNDATIONCOLLECTIONURI

Scope: Agent

The URI of the team foundation collection. For example: `https://fabrikamfiber.visualstudio.com/`.

System.TeamProject

Environment variable: SYSTEM_TEAMPROJECT

Scope: All

The name of the project that contains this build.

System.TeamProjectId

Environment variable: SYSTEM_TEAMPROJECTID

Scope: All

The ID of the project that this build belongs to.

TF_BUILD

Scope: Agent

Set to `True` if the script is being run by a build task.

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 Continuous integration (CI) triggers	The system identity, for example: `[DefaultCollection]\Project Collection Service Accounts`	The person who pushed or checked in the changes.
In Git or by a branch policy build.	The system identity, for example: `[DefaultCollection]\Project Collection Service Accounts`	The person who checked in the changes.
In TFVC by a gated check-in trigger	The person who checked in the changes.	The person who checked in the changes.
In Git or TFVC by the Scheduled triggers	The system identity, for example: `[DefaultCollection]\Project Collection Service Accounts`	The system identity, for example: `[DefaultCollection]\Project Collection Service Accounts`
Because you clicked the **Queue build** button	You	You

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?

7/17/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 CI/CD that helps your team **continuously deliver** software to your customers at a faster pace and with lower risk. You can **fully automate** the testing and delivery of your software in multiple 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 pipelines, configurations, and deployments. It also provides a history of all the activity performed during each deployment. Each release is accompanied by a listing of new features and developer commits that went into that release.

How does Release Management work?

The Release Management service stores the data about your release pipelines, 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 pipeline can contain settings to select an appropriate agent at runtime.

4. **Download artifacts**: The agent downloads all the artifacts specified in that release (provided you have not opted to skip the download). The agent currently understands two types of artifacts: Team Build artifacts and Jenkins artifacts.

5. **Run the deployment tasks**: The agent then runs all the tasks in the deployment job to deploy the app to the target servers for an 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. **Create your first build and release**

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 pipelines and release names

7/17/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A **release pipeline** is one of the fundamental concepts in Release Management for your DevOps CI/CD processes. 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 pipeline for your application. To author a release pipeline, 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 pipeline in shown below:

What's the difference between a release pipeline and a release?

In this example, a release of a website is created by collecting specific versions of two builds (artifacts), each from a different build pipeline. The release is first deployed to a Dev environment and then forked 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 app to websites across the globe in parallel within production ring 1. After all 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 pipelines have a few options that can be customized: release names and retention policies.

Release names

The names of releases for a release pipeline are, by default, sequentially numbered. The first release is named **Release-1**, the next release is **Release-2**, and so on. You can change this naming scheme by editing the release name format mask. In the **Options** tab of a release pipeline, edit the **Release name format** property.

When specifying the format mask, you can use the following pre-defined variables.

VARIABLE	DESCRIPTION
Rev:rr	An auto-incremented number with at least the specified number of digits.
Date / Date:MMddyy	The current date, with the default format **MMddyy**. Any combinations of M/MM/MMM/MMMM, d/dd/ddd/dddd, y/yy/yyyy/yyyy, h/hh/H/HH, m/mm, s/ss are supported.
System.TeamProject	The name of the project to which this build belongs.
Release.ReleaseId	The ID of the release, which is unique across all releases in the project.
Release.DefinitionName	The name of the release pipeline to which the current release belongs.
Build.BuildNumber	The number of the build contained in the release. If a release has multiple builds, this is the number of the primary build.
Build.DefinitionName	The pipeline name of the build contained in the release. If a release has multiple builds, this is the pipeline name of the primary build.

VARIABLE	DESCRIPTION
Artifact.ArtifactType	The type of the artifact source linked with the release. For example, this can be **Team Build** or **Jenkins**.
Build.SourceBranch	The branch of the primary artifact source. For Git, this is of the form **master** if the branch is **refs/heads/master**. For Team Foundation Version Control, this is of the form **branch** if the root server path for the workspace is **$/teamproject/branch**. This variable is not set for Jenkins or other artifact sources.
Custom variable	The value of a global configuration property defined in the release pipeline.

For example, the release name format `Release $(Rev:rrr) for build $(Build.BuildNumber) $(Build.DefinitionName)` will create releases with names such as **Release 002 for build 20170213.2 MySampleAppBuild**.

Release retention

You can customize how long releases of this pipeline must be retained. For more information, see release retention.

Release history

Every time you save a release pipeline, 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.

Release environments, queuing policies, and options

7/9/2018 • 8 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

An **environment** is a *logical* and *independent* entity that represents where you want to deploy a release generated from a release pipeline. We'll examine these two characteristics in more detail to help you understand how to divide your release process into environments in your DevOps CI/CD processes.

First, an environment in a release pipeline 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 pipeline. For example, your pipeline 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 pipeline.
- **Customer adoption rings** (for example, early adopter ring, frequent adopter ring, late adopter ring) - You typically want to deploy new or beta releases to your early adopters more often than to other users. Therefore, you are likely to have different releases in each of these rings. This is a good example of the use of environments in a pipeline.
- **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 pipelines, 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 pipeline 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 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 pipeline are created in quick succession due to new builds being produced rapidly. Assume that the first environment in this pipeline is named **QA** and has both pre-deployment and post-deployment approvers defined.

- If you do not specify a limit for the number of parallel deployments, all five approval requests will be sent out as soon as the releases are created. If the approvers grant approval for all of the releases, they will all be deployed to the **QA** 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 pipeline. 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.

Q & A

Why do we call these *environments* instead of *stages*?

Primarily, to communicate the intent that they must be independent in a release pipeline. We felt that this term implies that you can have two different releases (sets of build artifacts) deployed to two environments within a pipeline 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 pipeline 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 pipeline?

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 pipeline 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.

Deployment groups

7/17/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A deployment group is a logical set of deployment target machines that have agents installed on each one. Deployment groups represent the physical environments; for example, "Dev", "Test", "UAT", and "Production". In effect, a deployment group is just another grouping of agents, much like an agent pool.

When authoring a VSTS or TFS Release pipeline, 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 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 pipeline 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 pipeline or a release, use the links in the **Releases** tab.

Share a deployment group

Each deployment group is a member of a **deployment pool**, and you can share the deployment pool and groups across projects provided that:

- The user sharing the deployment pool has User permission for the pool containing the group.
- The user sharing the deployment pool has permission to create a deployment group in the project where it is being shared.
- The project does not already contain a deployment group that is a member of the same deployment pool.

The tags you assign to each machine in the pool are scoped at project level, so you can specify a different tag for the same machine in each deployment group.

Add a deployment pool and group to another project

To manage a deployment pool, or to add an existing deployment pool and the groups it contains to another project, choose the **Manage** link in the **Agent Pool** section of the **Deployment Group** page. In the **Deployment Pools** page, select the projects for which you want the deployment group to be available, then save the changes.

When you navigate to the **Deployment Groups** page in the target project(s), you will see the deployment group you added and you can assign project-specific machine tags as required.

Create a new deployment pool

You can add a new deployment pool, share it amongst your projects, and then add deployment groups to it. In the **Deployment Pools** page, choose **+ New**. In the **New deployment pool** panel, enter a name for the pool and then select the projects for which you want it to be available.

When you navigate to the **Deployment Groups** page in the target project(s), you will see the deployment group you added and you can assign project-specific machine tags as required.

Related topics

- Run on machine group 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.

How To: Provision agents for deployment groups

7/19/2018 • 6 minutes to read • Edit Online

VSTS | TFS 2018

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Deployment groups make it easy to define logical groups of target machines for deployment, and install the required agent on each machine. This topic explains how to create a deployment group, and install and provision the agent on each virtual or physical machine in your deployment group.

You can install the agent in any one of these ways:

- Run the script that is generated automatically when you create a deployment group.
- Install the **VSTS Agent** Azure VM extension on each of the VMs.
- Use the **Azure Resource Group Deployment** task in your release pipeline.

For information about agents and pipelines, see:

- Concurrent pipelines in Team Foundation Server.
- Concurrent jobs in VSTS.
- Pricing for VSTS features

Run the installation script on the target servers

1. In the **Deployment groups** tab of the **Build & Release** hub, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Register machines using command line** section of the next page, select the target machine operating system.
4. Choose **Use a personal access token in the script for authentication**. Learn more.
5. Choose **Copy the script to clipboard**.
6. Log onto each target machine in turn using the account with the appropriate permissions and:
   - Open an Administrator PowerShell command prompt, paste in the script you copied, then execute it to register the machine with this group.
   - If you get an error when running the script that a secure channel could not be created, execute this command at the Administrator PowerShell prompt:

     ```
     [Net.ServicePointManager]::SecurityProtocol = [Net.SecurityProtocolType]::Tls12
     ```

   - When prompted to configure tags for the agent, press `Y` and enter any tags you will use to identify subsets of the machines in the group for partial deployments.

     > Tags you assign allow you to limit deployment to specific servers when the deployment group is used in a **Run on machine group** phase.

- When prompted for the user account, press *Return* to accept the defaults.
- Wait for the script to finish with the message `Service vstsagent.{organization-name}.{computer-name} started successfully`.

7. In the **Deployment groups** page of the **Build & Release** hub, open the **Machines** tab and verify that the agents are running. If the tags you configured are not visible, refresh the page.

Install the VSTS Agent Azure VM extension

1. In the **Deployment groups** tab of the **Build & Release** hub, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the Azure portal, for each VM that will be included in the deployment group open the **Extension** blade, choose **+ Add** to open the **New resource** list, and select **VSTS Agent**.

4. In the **Install extension** blade, specify the name of the VSTS subacription to use. For example, if the URL is `https://contoso.visualstudio.com`, just specify **contoso**.
5. Specify the project name and the deployment group name.
6. Optionally, specify a name for the agent. If not specified, it uses the VM name appended with `-DG`.
7. Enter the Personal Access Token (PAT) to use for authentication against VSTS.
8. Optionally, specify a comma-separated list of tags that will be configured on the agent. Tags are not case-sensitive, and each must no more than 256 characters.
9. Choose **OK** to begin installation of the agent on this VM.
10. Add the extension to any other VMs you want to include in this deployment group.

Use the Azure Resource Group Deployment task

You can use the Azure Resource Group Deployment task to deploy an Azure Resource Manager (ARM) template that installs the VSTS Agent Azure VM extension as you create a virtual machine, or to update the resource group to apply the extension after the virtual machine has been created. Alternatively, you can use the advanced deployment options of the Azure Resource Group Deployment task to deploy the agent to deployment groups.

Install the "VSTS Agent" Azure VM extension using an ARM template

An ARM template is a JSON file that declaratively defines a set of Azure resources. The template can be automatically read and the resources provisioned by Azure. In a single template, you can deploy multiple services along with their dependencies.

For a Windows VM, create an ARM template and add a resources element under the `Microsoft.Compute/virtualMachine` resource as shown here:

```
"resources": [
  {
    "name": "[concat(parameters('vmNamePrefix'),copyIndex(),'/TeamServicesAgent')]",
    "type": "Microsoft.Compute/virtualMachines/extensions",
    "location": "[parameters('location')]",
    "apiVersion": "2015-06-15",
    "dependsOn": [
        "[resourceId('Microsoft.Compute/virtualMachines/',
                      concat(parameters('vmNamePrefix'),copyindex()))]"
    ],
    "properties": {
      "publisher": "Microsoft.VisualStudio.Services",
      "type": "TeamServicesAgent",
      "typeHandlerVersion": "1.0",
      "autoUpgradeMinorVersion": true,
      "settings": {
        "VSTSAccountName": "[parameters('VSTSAccountName')]",
        "TeamProject": "[parameters('TeamProject')]",
        "DeploymentGroup": "[parameters('DeploymentGroup')]",
        "AgentName": "[parameters('AgentName')]",
        "Tags": "[parameters('Tags')]"
      },
      "protectedSettings": {
      "PATToken": "[parameters('PATToken')]"
     }
   }
  }
]
```

where:

- **VSTSAccountName** is required. The VSTS subscription to use. Example: If your URL is `https://contoso.visualstudio.com`, just specify `contoso`
- **TeamProject** is required. The project that has the deployment group defined within it
- **DeploymentGroup** is required. The deployment group against which deployment agent will be registered
- **AgentName** is optional. If not specified, the VM name with `-DG` appended will be used
- **Tags** is optional. A comma-separated list of tags that will be set on the agent. Tags are not case sensitive and each must be no more than 256 characters
- **PATToken** is required. The Personal Access Token that will be used to authenticate against VSTS to download and configure the agent

> **Note**: If you are deploying to a Linux VM, ensure that the `type` parameter in the code is `TeamServicesAgentLinux`.

For more information about ARM templates, see Define resources in Azure Resource Manager templates.

To use the template:

1. In the **Deployment groups** tab of the **Build & Release** hub, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Releases** tab of the **Build & Release** hub, create a release pipeline with an environment that contains the **Azure Resource Group Deployment** task.
4. Provide the parameters required for the task such as the Azure subscription, resource group name, location, and template information, then save the release pipeline.
5. Create a release from the release pipeline to install the agents.

Install agents using the advanced deployment options

1. In the **Deployment groups** tab of the **Build & Release** hub, choose **+New** to create a new group.
2. Enter a name for the group, and optionally a description, then choose **Create**.
3. In the **Releases** tab of the **Build & Release** hub, create a release pipeline with an environment that contains the **Azure Resource Group Deployment** task.
4. Select the task and expand the **Advanced deployment options for virtual machines** section. Configure the parameters in this section as follows:
   - **Enable Prerequisites**: select **Configure with Deployment Group Agent**.
   - **TFS/VSTS endpoint**: Select an existing Team Foundation Server/TFS service connection that points to your target. Agent registration for deployment groups requires access to your Visual Studio project. If you do not have an existing service connection, choose **Add** and create one now. Configure it to use a Personal Access Token (PAT) with scope restricted to **Deployment Group**.
   - **Project**: Specify the project containing the deployment group.
   - **Deployment Group**: Specify the name of the deployment group against which the agents will be registered.
   - **Copy Azure VM tags to agents**: When set (ticked), any tags already configured on the Azure VM will be copied to the corresponding deployment group agent. By default, all Azure tags are copied using the format `Key: Value`. For example, `Role: Web`.

5. Provide the other parameters required for the task such as the Azure subscription, resource group name,

and location, then save the release pipeline.

6. Create a release from the release pipeline to install the agents.

Related topics

- Run on machine group 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.

Release artifacts and artifact sources

7/17/2018 • 18 minutes to read • Edit Online

[!INCLUDE [ver

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A release is a collection of artifacts in your DevOps CI/CD processes. An **artifact** is a deployable component of your application. 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 pipeline**, you link the appropriate **artifact sources** to your release pipeline. For example, you might link a Team Build build pipeline or a Jenkins project to your release pipeline.

When **creating a release**, you specify the exact version of these artifact sources; for example, the number of a build coming from 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 pipeline can be linked to **multiple artifact sources**, of which one is the primary source. In this case, when you create a release, you specify individual versions for each of these sources.

Artifacts are central to a number of features in Release Management. Some of the features that depend on the linking of artifacts to a release pipeline are:

- **Auto-trigger releases**. You can configure new releases to be automatically created whenever a new version of an artifact is produced. For more details, see Continuous deployment triggers. Note that the ability to automatically create releases is available for only some artifact sources.
- **Trigger conditions**. You can configure a release to be created automatically, or the deployment of a release to an environment to be triggered automatically, when only specific conditions on the artifacts are

met. For example, you can configure releases to be automatically created only when a new build is produced from a certain branch.

- **Artifact versions**. You can configure a release to automatically use a specific version of the build artifacts, to always use the latest version, or to allow you to specify the version when the release is created.
- **Artifact variables**. Every artifact that is part of a release has metadata associated with it, exposed to tasks through variables. This metadata includes the version number of the artifact, the branch of code from which the artifact was produced (in the case of build or source code artifacts), the pipeline that produced the artifact (in the case of build artifacts), and more. This information is accessible in the deployment tasks. For more details, see Artifact variables.
- **Work items and commits**. The work items or commits that are part of a release are computed from the versions of artifacts. For example, each build in 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, by default 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. Or you can use repositories such as Package Management in Visual Studio Team Services or a NuGet repository to store your artifacts. You can configure Release Management to deploy artifacts from all these sources.

By default, a release created from the release pipeline will use the latest version of the artifacts. At the time of linking an artifact source to a release pipeline, you can change this behavior by selecting one of the options to use the latest build from a specific branch by specifying the tags, a specific version, or allow the user to specify the version when the release is created from the pipeline.

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
- TFVC, Git, and GitHub
- Jenkins
- Azure Container Registry, Docker, and Kubernetes
- NuGet, npm, and Maven with Package Management
- External or on-premises TFS
- TeamCity
- Other sources

Team Build

You can link a release pipeline to any of the build pipelines in VSTS or TFS project collection.

> **NOTE**
>
> You must include a **Publish Artifacts** task in your build pipeline. For XAML build pipelines, an artifact with the name **drop** is published implicitly.

Some of the differences in capabilities between different versions of TFS and VSTS are:

- **TFS 2015**: You can link build pipelines only from the same project of your collection. You can link multiple definitions, but you cannot specify default versions. You can set up a continuous deployment trigger on only one of the definitions. When multiple build pipelines are linked, the latest builds of all the other definitions are used, along with the build that triggered the release creation.
- **TFS 2017 and newer** and **VSTS**: You can link build pipelines from any of the projects in VSTS or TFS. You can link multiple build pipelines and specify default values for each of them. You can set up continuous deployment triggers on multiple build sources. When any of the builds completes, it will trigger the creation of a release.

The following features are available when using 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 Continuous Deployment for details. You do not need to configure anything within the build pipeline. See the notes above for differences between version of TFS.
Artifact variables	A number of artifact variables are supported for builds from 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 skip the download of artifacts.
Deployment section in build	The build summary includes a **Deployment** section, which lists all the environments to which the build was deployed.

TFVC, Git, and GitHub

There are scenarios in which you may want to consume artifacts stored in a version control system directly, without passing them through a build 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 pipeline, you can link both a build pipeline that produces the binaries of the application as well as a version control repository that stores the configuration files into the same pipeline, and use the two sets of artifacts together while deploying.

Release Management integrates with Team Foundation Version Control (TFVC) repositories, Git repositories, and GitHub repositories.

You can link a release pipeline to any of the Git or TFVC repositories in any of the projects in your collection (you will need read access to these repositories). No additional setup is required when deploying version control artifacts within the same collection.

When you link a **Git** or **GitHub** repository and select a branch, you can edit the default properties of the artifact types after the artifact has been saved. This is particularly useful in scenarios where the branch for the stable version of the artifact changes, and continuous delivery releases should use this branch to obtain newer versions of the artifact. You can also specify details of the checkout, such as whether check out submodules and LFS-tracked files, and the shallow fetch depth.

When you link a **TFVC branch**, you can specify the changeset to be deployed when creating a release.

The following features are available when using TFVC, Git, and GitHub sources:

FEATURE	BEHAVIOR WITH TFVC, GIT, AND GITHUB SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for pushes into the repository in a release pipeline. This can automatically trigger a release when a new commit is made to a repository. See Triggers.
Artifact variables	A number of artifact variables are supported for version control sources.
Work items and commits	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 skip the download of artifacts.

Jenkins

To consume Jenkins artifacts, you must create a service connection with credentials to connect to your Jenkins server. For more details, see service connections and Jenkins service connection. You can then link a Jenkins project to a release pipeline. The Jenkins project must be configured with a post build action to publish the artifacts.

The following features are available when using Jenkins sources:

FEATURE	BEHAVIOR WITH JENKINS SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for pushes into the repository in a release pipeline. This can automatically trigger a release when a new commit is made to a repository. See Triggers.
Artifact variables	A number of artifact variables are supported for builds from Jenkins.
Work items and commits	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 skip the download of artifacts.

Artifacts generated by Jenkins builds are typically propagated to storage repositories for archiving and sharing. Azure blob storage is one of the supported repositories, allowing you to consume Jenkins projects that publish to Azure storage as artifact sources in a release pipeline. Deployments download the artifacts automatically from Azure to the agents. In this configuration, connectivity between the agent and the Jenkins server is not required. Microsoft-hosted agents can be used without exposing the server to internet.

> **NOTE**
>
> 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.

Azure Container Registry, Docker, Kubernetes

When deploying containerized apps, the container image is first pushed to a container registry. After the push is complete, the container image can be deployed to the Web App for Containers service or a Docker/Kubernetes cluster. You must create a service connection with credentials to connect to your service to deploy images located there, or to Azure. For more details, see service connections.

The following features are available when using Azure Container Registry, Docker, Kubernetes sources:

FEATURE	BEHAVIOR WITH DOCKER SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for images. This can automatically trigger a release when a new commit is made to a repository. See Triggers.
Artifact variables	A number of artifact variables are supported for builds.
Work items and commits	Release Management cannot show work items or commits.
Artifact download	By default, builds are downloaded to the agent. You can configure an option in the environment to skip the download of artifacts.

NuGet, npm, and Maven with Package Management

To integrate with NuGet, npm, or Maven, you must first install the Package Management extension from Marketplace. For more information, see Package Management Overview.

Scenarios where you may want to consume Package Management artifacts are:

1. You have your application build (such as TFS, VSTS, TeamCity, Jenkins) published as a package (NuGet, npm, or Maven) to Package Management and you want to consume the artifact in a release.
2. As part of your application deployment, you need additional packages stored in Package Management.

When you link a Package Management artifact to your release pipeline, you must select the Feed, Package, and the Default version for the package. You can choose to pick up the latest version of the package, use a specific version, or select the version at the time of release creation. During deployment, the package is downloaded to the agent folder and the contents are extracted as part of the phase execution.

The following features are available when using Package Management sources:

FEATURE	BEHAVIOR WITH PACKAGE MANAGEMENT SOURCES
Auto-trigger releases	You can configure a continuous deployment trigger for packages. This can automatically trigger a release when a package is updated. See Triggers.
Artifact variables	A number of artifact variables are supported for packages.
Work items and commits	Release Management cannot show work items or commits.
Artifact download	By default, packages are downloaded to the agent. You can configure an option in the environment to skip the download of artifacts.

External or on-premises 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 subscription from another VSTS, 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 connection with credentials to connect to your TFS server (see service connections for details).

You can then link a TFS build pipeline to your release pipeline. Choose **External TFS Build** in the **Type** list.

The following features are available when using external TFS sources:

FEATURE	BEHAVIOR WITH EXTERNAL TFS SOURCES
Auto-trigger releases	You cannot configure a continuous deployment trigger for external TFS sources in a release pipeline. To automatically create a new release when a build is complete, you would need to add a script to your build pipeline in the external TFS server to invoke REST APIs of Release Management and to create a new release.
Artifact variables	A number of artifact variables 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 skip the download 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 pipelines 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.

TeamCity

To integrate with TeamCity, you must first install the TeamCity artifacts for Release Management extension from Marketplace.

To consume TeamCity artifacts, start by creating a service connection with credentials to connect to your TeamCity server (see service connections for details).

You can then link a TeamCity build configuration to a release pipeline. The TeamCity build configuration must be configured with an action to publish the artifacts.

The following features are available when using TeamCity sources:

FEATURE	BEHAVIOR WITH TEAMCITY SOURCES
Auto-trigger releases	You cannot configure a continuous deployment trigger for TeamCity sources in a release pipeline. To create a new release automatically when a build is complete, add a script to your TeamCity project that invokes the Release Management REST APIs to create a new release.
Artifact variables	A number of artifact variables 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 skip the download 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.

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 pipeline, 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 pipeline when you initiate a release, and the artifacts are downloaded into that folder. The `$(System.DefaultWorkingDirectory)` variable maps to this folder.

Note that, at present, 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.

In VSTS, you can, however, select which artifacts you want to download to the agent for a specific phase and environment of the deployment. Typically, you will do this to improve the efficiency of the deployment process when the tasks in that phase do not require all or any of the 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 pipeline is automatically provided with a specific download location known as the *source alias*. This location can be accessed through the variable:

```
$(System.DefaultWorkingDirectory)\[source alias]
```

This uniqueness also ensures that, if you later rename a linked artifact source in its original location (for example, rename a build pipeline in 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, prefixed with an underscore; depending on the type of the artifact source this will be the name of the build pipeline, job, project,

or repository. You can edit the source alias from the artifacts tab of a release pipeline; for example, when you change the name of the build pipeline and you want to use a source alias that reflects the name of the build pipeline.

> The source alias can contain only alphanumeric characters and underscores, and must start with a letter or an underscore

Primary source

When you link multiple artifact sources to a release pipeline, one of them is designated as the primary artifact source. The primary artifact source is used to set a number of pre-defined variables. It can also be used in naming releases.

Artifact variables

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 pipeline, you can access information about each of these. For a list of all pre-defined artifact variables, see variables.

Contributed links and additional information

- Code repo sources in Release Management
- TeamCity extension for Release Management
- Jenkins artifacts in Release Management
- External TFS artifacts for Release Management

Related topics

- Release pipelines
- 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.

Release, branch, and environment triggers

7/17/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can configure when releases should be created, and when those releases should be deployed to environments, in your DevOps CI/CD processes. The former is configured through release triggers, and the latter through environment triggers - both in a release pipeline.

Release (continuous deployment) triggers

If you specify certain types of artifacts in a release pipeline, 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.

If you have linked multiple Team Foundation Build artifacts to a release pipeline, you can configure continuous deployment for each of them. In other words, you can choose to have a release created automatically when a new build of any of those artifacts is produced.

You add build branch filters if you want to create the release only when the build is produced by compiling code from certain branches (only applicable when the code is in a TFVC, Git, or GitHub repository) or when the build has certain tags. These can be both include and exclude filters. For example, use **features/*** to include all builds under the **features** branch. You can also include custom variables in a filter value.

Alternatively, you can specify a filter to use the default branch specified in the build pipeline. This is useful when, for example, the default build branch changes in every development sprint. It means you don't need to update the trigger filter across all release pipelines for every change - instead you just change the default branch in the build pipeline.

> Note that, even though a release is automatically created, it might not be deployed automatically to any 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 pipeline, 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.

- **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 pipeline. 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.

- **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 pipeline 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 pipeline, 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 pipeline 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.

Default and custom release variables and debugging

7/17/2018 • 11 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

As you compose the tasks for deploying your application into each environment in your DevOps CI/CD processes, 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 view the current values of all variables for a release, and use a default variable to run a release in debug mode.

Custom variables

Custom variables can be defined at various scopes.

- Share values across all of the definitions in a project by using variable groups. Choose a variable group when you need to use the same values across all the definitions, 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 pipeline variables**. Choose a release pipeline variable when you need to use the same value across all the environments and tasks in the release pipeline, and you want to be able to change the value in a single place. You define and manage these variables in the **Variables** tab in a release pipeline.
- Share values across all of the tasks within one specific 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 pipeline.

Using custom variables at project, release pipeline, 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 pipelines. Designate a configuration property to be a secure (secret) variable by selecting the 🔒 (padlock) icon next to the variable.

> The values of hidden (secret) variables are stored securely on the server and cannot be viewed by users

> after they are saved. During a deployment, the Release Management service decrypts these values when referenced by the tasks and passes them to the agent over a secure HTTPS channel.

Using custom variables

To use custom variables in your build and release tasks, simply enclose the variable name in parentheses and precede it with a **$** character. For example, if you have a variable named **adminUserName**, you can insert the current value of that variable into a parameter of a task as `$(adminUserName)`.

> **Note:** At present, variables in different groups that are linked to a pipeline in the same scope (such as a release or environment scope) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

You can use custom variables to prompt for values during the execution of a release. For more details, see Approvals.

Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command. Note that the updated variable value is scoped to the phase being executed, and does not flow across phases or environments.

> **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. Some of the most significant variables are described in the following tables.

System variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.TeamFoundationServerUri	The URL of the Release Management service connection in TFS or VSTS. Use this from your scripts or tasks to call REST APIs on the Release Management service.	https://fabrikam.vsrm.visualstudio.com/	
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or VSTS. 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 project to which this build or release belongs.	Fabrikam	
System.TeamProjectId	The ID of the project to which this build or release belongs.	79f5c12e-3337-4151-be41-a268d2c73344	TFS 2015
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	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.DefaultWorkingDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.ArtifactsDirectory.	C:\agent_work\r1\a	
System.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.	C:\agent_work	
System.Debug	This is the only system variable that can be *set* by the users. Set this to true to run the release in debug mode to assist in fault-finding.	true	

Release variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.DefinitionName	The name of the release pipeline to which the current release belongs.	fabrikam-cd	
Release.DefinitionId	The ID of the release pipeline to which the current release belongs.	1	TFS 2015
Release.ReleaseName	The name of the current release.	Release-47	
Release.ReleaseId	The identifier of the current release record.	118	
Release.ReleaseUri	The URI of current release.	vstfs:///ReleaseManagement/Release/118	
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	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
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 pipeline.	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 agent pool. This is likely to be different from the computer name.	fabrikam-agent	
Agent.MachineName	The name of the computer on which the agent is configured.	fabrikam-agent	
Agent.Version	The version of the agent software.	2.109.1	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
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	
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

General artifact variables

For each artifact that is referenced in a release, you can use the following artifact variables. Not all variables are meaningful for each artifact type. The table below lists the default artifact variables and provides examples of the values that they have depending on the artifact type. If an example is empty, it implies that the variable is not populated for that artifact type.

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/ TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.DefinitionId	The identifier of the build pipeline or repository.	1			fabrikam/asp

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/ TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.DefinitionName	The name of the build pipeline or repository.	fabrikam-ci		TFVC: $/fabrikam, Git: fabrikam	fabrikam/asp (master)
Release.Artifacts.{alias}.BuildNumber	The build number or the commit identifier.	20170112.1	20170112.1	TFVC: Changeset 3, Git: 38629c964	38629c964
Release.Artifacts.{alias}.BuildId	The build identifier.	130	130		38629c964d21fe405ef830b7d0220966b82c9e11
Release.Artifacts.{alias}.BuildURI	The URL for the build.	vstfs:///build-release/Build/130			
Release.Artifacts.{alias}.SourceBranch	The path of the branch from which the source was built.	refs/heads/master			
Release.Artifacts.{alias}.SourceBranchName	The name of the branch from which the source was built.	master			
Release.Artifacts.{alias}.SourceVersion	The commit that was built.	bc0044458ba1d9298cdc649cb5dcf013180706f7			
Release.Artifacts.{alias}.Repository.Provider	The type of repository from which the source was built	Git			
Release.Artifacts.{alias}.RequestedForID	The identifier of the account that triggered the build.	2f435d07-769f-4e46 -849d-10d1ab9ba6ab			
Release.Artifacts.{alias}.RequestedFor	The name of the account that requested the build.	Mateo Escobedo			
Release.Artifacts.{alias}.Type	The type of artifact source, such as Build.	Build	Jenkins: Jenkins, TeamCity:TeamCity	TFVC: TFVC, Git: Git	GitHub

See also Artifact source alias

Primary artifact variables

You designate one of the artifacts as a primary artifact in a release pipeline. For the designated primary artifact, 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
Build.BuildNumber	Release.Artifacts.{Primary artifact alias}.BuildNumber
Build.BuildId	Release.Artifacts.{Primary artifact alias}.BuildId
Build.BuildURI	Release.Artifacts.{Primary artifact alias}.BuildURI
Build.SourceBranch	Release.Artifacts.{Primary artifact alias}.SourceBranch
Build.SourceBranchName	Release.Artifacts.{Primary artifact alias}.SourceBranchName
Build.SourceVersion	Release.Artifacts.{Primary artifact alias}.SourceVersion
Build.Repository.Provider	Release.Artifacts.{Primary artifact alias}.Repository.Provider
Build.RequestedForID	Release.Artifacts.{Primary artifact alias}.RequestedForID
Build.RequestedFor	Release.Artifacts.{Primary artifact alias}.RequestedFor
Build.Type	Release.Artifacts.{Primary artifact alias}.Type

Using default variables

You can use the default variables in two ways - as parameters to tasks in a release pipeline or in your scripts.

You can directly use a default variable as an input to a task. For example, to pass `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** to a task, you would use `$(Release.Artifacts.ASPNET4.CI.DefinitionName)`.

To use a default variable in your script, you must first replace the `.` in the default variable names with `_`. For example, to print the value of artifact variable `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** in a Powershell script, you would use `$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME`.

Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

View the current values of all release variables

In the **Logs** page of a release summary, choose the **Initialize Job** step. The log pane includes a list of all the variables and their values for this release.

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 pipeline.
- 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 pipeline.

> If you get an error related to an Azure RM service connection, see How to: Troubleshoot Azure Resource Manager service connections.

Help and support

- See our troubleshooting page.

- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Release approvals and gates overview

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A release pipeline specifies the end-to-end release 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 pipeline can be configured with pre-deployment and post-deployment conditions that can include waiting for users to manually approve or reject deployments, and checking with other automated systems until specific conditions are verified. In addition, you can configure a manual intervention to pause the deployment 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 pipeline.

By using approvals, gates, and manual intervention you can take full control of your releases to meet a wide range of deployment requirements. Typical scenarios where approvals, gates, and manual intervention are useful include the following.

SCENARIO	FEATURE(S) TO USE
Some users must manually validate the change request and approve the deployment to an environment.	Pre-deployment approvals

SCENARIO	FEATURE(S) TO USE
Some users must manually sign off the app after deployment before the release is promoted to other environments.	Post-deployment approvals
You want to ensure there are no active issues in the work item or problem management system before deploying a build to an environment.	Pre-deployment gates
You want to ensure there are no incidents from the monitoring or incident management system for the app after it's been deployed, before promoting the release.	Post-deployment gates
After deployment you want to wait for a specified time before prompting some users for a manual sign-off.	Post-deployment gates and post-deployment approvals
During the deployment process a user must manually follow specific instructions and then resume the deployment.	Manual Intervention
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.	Manual Intervention
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 pipeline to fully achieve your own process and deployment requirements.

Related topics

- Approvals
- Gates
- Manual intervention
- Environments
- Triggers
- Release pipelines 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.

Release deployment control using approvals

7/9/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

When a release is created from a release pipeline that defines approvals, the deployment stops at each point where approval is required until the specified approver grants approval or rejects the release (or re-assigns the approval to another user). You can enable manual deployment approvals for each environment in a release pipeline.

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.

   For a **post-deployment** approval, choose the icon at the exit point of the environment and enable post-deployment approvers.

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.

Approval notifications

Release Management can send notifications such as an email message to the approver(s) defined for each approval step. Configure recipients and settings in the **Notifications** page of the **Settings** hub.

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.

A notification bar is also shown in the release details page, with a link that displays the approval dialog.

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.

Release deployment control using gates

7/9/2018 • 5 minutes to read • Edit Online

VSTS

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

Scenarios for gates

Some scenarios and use cases for gates are:

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

Most of the health parameters vary over time, regularly changing their status from healthy to unhealthy and back to healthy. To account for such variations, all the gates are periodically re-evaluated until all of them are successful at the same time. The release execution and deployment does not proceed if all gates do not succeed in the same interval and before the configured timeout.

Define a gate for an environment

1. Enable gates in the **Pre-deployment conditions** or **Post-deployment conditions** panel for an environment.

2. Specify the required **Delay before evaluation**. This is a time delay at the beginning of gate evaluation process that allows the gates to initialize, stabilize, and begin providing accurate results for the current deployment. See Gate evaluation flows.

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 of the time taken for the deployed app to reach a steady operational state, the time taken for execution of all the required tests on the deployed environment, and the time it takes for incidents to be logged after the deployment.

3. Choose **+ Add** and select the type of release gate you require.

The following gates are provided by default:

- **Invoke Azure function**: Trigger execution of an Azure function and ensure a successful completion. For more details, see Azure function task.
- **Query Azure monitor alerts**: Observe the configured Azure monitor alert rules for active alerts. For more details, see Azure monitor task.
- **Invoke REST API**: Make a call to a REST API and continue if it returns a successful response. For more details, see HTTP REST API task.
- **Query Work items**: Ensure the number of matching work items returned from a query is within a threshold. For more details, see Work item query task.

You can create your own gates with Marketplace extensions.

4. Select and enter the required gate arguments, depending on the type of gate you chose.

5. Set the evaluation options that apply to all the gates you've added:

- **Time between re-evaluation of gates**. The time interval between successive evaluations of the gates. At each sampling interval, new requests are sent concurrently to each gate and the new results are evaluated. It is recommended that the sampling interval is greater than the longest typical response time of the configured gates to allow time for all responses to be received for evaluation.

- **Timeout after which gates fail**. The maximum evaluation period for all gates. The deployment will be rejected if the timeout is reached before all gates succeed during the same sampling interval.

- **Gates and approvals**. Select the required order of execution for gates and approvals if you have configured both. For pre-deployment conditions, the default is to prompt for manual (user) approvals first, then evaluate gates afterwards. This saves the system from evaluating the gate functions if the release is rejected by the user. For post-deployment conditions, the default is to evaluate gates and prompt for manual approvals only when all gates are successful. This ensures the approvers have all the information required for a sign-off.

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.

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.

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
- Tutorial: Use approvals and gates to control your deployment
- Twitter sentiment as a release gate
- GitHub issues as a release gate
- Author custom gates. Library with examples

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Videos

Environment templates in Release Management

7/19/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

When you start a new release pipeline, or when you add an environment to an existing release pipeline, 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 pipeline for your DevOps CI/CD processes.

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 pipeline or add a new environment to a pipeline. 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 pipelines.

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 pipelines (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 subscriptions, enterprises, or projects?

Custom templates that you create are scoped to the project that you created them in. Templates cannot be exported or shared with another project, collection, server, or organization. You can, however, export a release pipeline and import it into another project, collection, server, or subscription. Then you can re-create the template for use in that location.

Can I publish or consume new templates through extensions in VS Marketplace?

Yes. See Adding release 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 pipeline.

How do I update a custom environment template?

To update an environment template, delete the existing template in a release pipeline 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

7/9/2018 • 2 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 pipeline in your DevOps CI/CD processes. It includes a snapshot of all the information required to carry out all the tasks and actions in the release pipeline, such as the environments, the tasks for each one, the values of task parameters and variables, and the release policies such as triggers, approvers, and release queuing options. There can be multiple releases from one released pipeline, 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 pipeline. 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 pipelines, releases, and deployments.

Releases (and, in some cases, draft releases) can be created from a release pipeline in several ways:

- By a continuous deployment trigger that creates a release when a new version of the source build artifacts is available.
- By using the **Release** command in the UI to create a release manually from the Releases or the Builds summary.
- By sending a command over the network to the REST interface.

However, the action of creating a release ***does not*** mean it will automatically or immediately start a deployment. For example:

- There may be deployment triggers defined for 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 builds and releases

7/17/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

A **task** is the building block for defining automation in a build pipeline, or in an environment of a release pipeline. 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 subscription or collection, extends the task catalog with one or more tasks. Furthermore, you can write your own custom extensions to add tasks to VSTS or TFS.

When you add a task to your build or release pipeline, it may also add a set of **demands** to the pipeline. The demands define the prerequisites that must be installed on the agent for the task to run. When you run the build or deployment, an agent that meets these demands will be chosen.

When you queue a build or a deployment, all the tasks are run in sequence, one after the other, on an agent. To run the same set of tasks in parallel on multiple agents, or to run some tasks without using an agent, see 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 pipeline and manually change to the new major version. The build or release log will include an alert that a new major version is available.

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 pipeline change from the **History** tab. The ability to restore to an older version of a release pipeline is not currently available. You must manually revert the changes to the release pipeline, then save the pipeline.
- Consider cloning the pipeline and testing the cloned pipeline 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 pipeline 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.

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 Microsoft-hosted agents) just in time for your CI build.
- Validate your app or library against multiple versions of a dependency such as Node.js.

For example, you can set up your build 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 pipeline (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

7/17/2018 • 9 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can organize your build or deployment process into phases. Every build or deployment process has at least one phase.

> **NOTE**
> You must install TFS 2018.2 to use phases in build processes. In TFS 2018 RTM you can use phases in release management deployment processes.

You can organize your deployment process into phases. Every deployment process has at least one phase.

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

> **NOTE**
> You must install Update 2 to use phases in TFS 2017, and they are available only in release management deployment processes. Phases in build pipelines are available in VSTS, TFS 2018.2, and newer versions.

In a build process, the most common target is an agent. The other kind of target is the VSTS server.

In a build process, the most common target is an agent. The other kind of target is the server (your TFS instance).

In a deployment process, the target can be either an agent, a deployment group, or the server.

When the target is an agent, the tasks are run on the computer that hosts the agent.

- Web
- YAML

When you create a new pipeline, it starts with a single agent phase. The properties for the agent phase are displayed when you select the agent phase in the editor.

Demands

Use demands to specify what capabilities an agent must have to run jobs from your phase.

- Web
- YAML

You have the option to specify demands in the pipeline, in the phases, or both. If you specify demands in both the pipeline and in a phase, the union of the two sets of demands is required for the system to select an agent.

Learn more about build and release agent capabilities.

Container image

If you are using YAML, you can specify a Docker container to use for your agent phase.

- Web
- YAML

Containers are not yet supported in the web editor.

Timeouts

To avoid hanging up your resources when your process is hung or waiting too long, it's a good idea to set a limit on how long your process is allowed to run. Use the phase timeout setting to specify the limit in minutes for jobs run by this phase. A zero value means that the jobs will run for an effectively unlimited amount of time.

- Web
- YAML

Select the phase and then specify the timeout value.

On the **Options** tab you can specify default values for all phases in the pipeline. If you specify a non-zero value for the phase timeout, then it overrides any value that is specified in the pipeline options. If you specify a zero value, then the timeout value from the pipeline options is used. If the pipeline value is also set to zero, then there is no timeout.

> Jobs targeting Microsoft-hosted agents have additional restrictions on how long they may run.

> You can also set the timeout for each task individually - see task control options.

Multi-configuration

From a single phase you can run multiple jobs and multiple agents in parallel. Some examples include:

- **Multi-configuration builds:** An agent phase can be used in a build pipeline to build multiple configurations in parallel. For example, you could build a Visual C++ app for both `debug` and `release` configurations on both `x86` and `x64` platforms. To learn more, see Visual Studio Build - multiple configurations for multiple platforms.
- **Multi-configuration deployments:** An agent phase can be used in an environment of a release pipeline to run multiple deployment jobs in parallel, for example, to different geographic regions.
- **Multi-configuration testing:** An agent phase can be used in a build pipeline or in an environment of a release pipeline to run a set of tests in parallel - once for each test configuration.

- Web
- YAML

To run multiple jobs using multi-configuration option, 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. To use multipliers for build or deployment, you must:

- Define one or more variables on the **Variables** tab of the pipeline 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 subscription, 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`

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. The test slicing example above can be accomplished through multi-agent option.

Slicing

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 slicing, 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. Variables `System.SliceNumber` and `System.SliceCount` are added to each job.

- Web
- YAML

Specify the **multi-agent** option on an agent phase to leverage slicing. The job is dispatched as many times as the number of agents you specify, and the variables `System.SliceNumber` and `System.SliceCount` are automatically set in each job.

Phase variables

If you are using YAML, variables can be specified on the phase. The variables can be passed to task inputs using the macro syntax $(variableName), or accessed within a script using the environment variable.

- Web
- YAML

Phase variables are not yet supported in the web editor.

Artifact download

- Web
- YAML

In a release pipeline, 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.

Alternatively, You can choose to download specific artifacts during the job execution in a release. Use this option if the tasks in a particular phase rely on only specific artifacts.

Access to OAuth token

You can allow tasks running in this phase to access current VSTS or TFS OAuth security token. The token can be use to authenticate to the VSTS REST API.

- Web
- YAML

Select the **Allow scripts to access OAuth token** option in the control options for the phase.

Related topics

- Multiple phases
- Server phases
- Deployment group phases

7/9/2018 • 3 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Server phases

Tasks in a server phase are orchestrated by and executed on the server (VSTS or TFS). A server phase does not require an agent or any target computers. Only a few tasks, such as the Manual Intervention and Invoke REST API tasks, are supported in a server phase at present. At present you can add only one task to each server phase in your pipeline.

- Web
- YAML

You add a server phase in the editor by selecting '...' on **Process** channel in the **Tasks** tab of a pipeline. The properties for the server phase are displayed when you select the phase in the editor.

Timeouts

Use the phase timeout 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. Jobs targeting Microsoft-hosted agents have additional restrictions on how long they may run.

- Web
- YAML

If you specify a non-zero value for the phase timeout, then it overrides any value that is specified in the pipeline options. If you specify a zero value, then the timeout value from the pipeline options is used. If the pipeline value is also set to zero, then there is no timeout.

Multi-configuration

If you are using YAML, you can dispatch multiple server jobs from a single server phase.

- Web
- YAML

Multi-configuration server phases are not supported in web designer.

Phase variables

If you are using YAML, variables can be specified on the phase. The variables can be passed to task inputs using the macro syntax $(variableName), or accessed within a script using the environment variable.

- Web
- YAML

Phase variables are not yet supported in the web editor.

Related topics

- Phases
- Multiple phases
- Deployment group phases

7/9/2018 • 2 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Deployment group phases

Deployment groups make it easy to define groups of target servers for deployment. 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. You can also 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.

- Web
- YAML

You add a deployment group phase in the editor by selecting '...' on **Process** channel in the **Tasks** tab of a release pipeline. The properties for the deployment group phase are displayed when you select the phase in the editor.

> **NOTE**
>
> Deployment group phases can only be used in release pipelines. They cannot be used in build pipelines.

Timeouts

Use the phase timeout 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. Jobs targeting Microsoft-hosted agents have additional restrictions on how long they may run.

Related topics

- Phases
- Server phases
- Multiple phases

Multiple phases

7/9/2018 • 5 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can add multiple phases to a pipeline. By using different phases in a pipeline, you can:

- Partition your process into sections targeting different agent queues
- Partition your process into sections targeting different sets of self-hosted agents using different demands
- Partition your process into sections that run on agents and those that run without an agent
- Publish build artifacts in one phase and consume those in subsequent phases
- Pause the deployment in the middle using a manual intervention task
- Reduce build time by running multiple phases in parallel
- Reduce deployment time by selectively downloading different artifacts in different phases of a deployment process
- Conditionally execute a set of tasks

- Partition your process into sections targeting different agent queues
- Partition your process into sections targeting different sets of self-hosted agents using different demands
- Partition your process into sections that run on agents and those that run without an agent
- Pause the deployment in the middle using a manual intervention task

> **NOTE**
> Phases are not supported in build pipelines in TFS 2017.

- Web
- YAML

To add a new phase, select '...' on the process channel in **Tasks** tab of the pipeline. The conditions and order of execution for a phase are displayed when you select the phase in the editor.

Dependencies

- Web
- YAML

When you specify multiple phases in a build pipeline, they run in parallel by default. You can specify the order in which phases must execute by configuring dependencies between phases. Phase dependencies are not yet supported in release pipelines. Multiple phases in a release pipeline run in sequence.

Multiple phases you add to a build pipeline run in parallel. Multiple phases you add to a release pipeline run in sequence. You cannot configure the order of dependencies between phases in this version of TFS.

Multiple phases you add to a release pipeline run in sequence. You cannot configure the order of dependencies between phases in this version of TFS. You cannot also use phases with build pipelines.

For example, the pipeline shown below divides the overall release execution into separate execution phases by using two agent phases and a server phase.

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 server 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. 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.

Conditions

You can specify the conditions under which each phase runs. By default, a phase runs if it does not depend on any other phase, or if all of the phases that it depends on have completed and succeeded. You can customize this behavior by forcing a phase to run even if a previous phase fails or by specifying a custom condition.

- Web
- YAML

Use the **Run this phase** option on an agent or server phase to run the tasks 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 pipeline.

Related topics

- Phases
- Server phases
- Deployment group phases

Specify conditions

7/9/2018 • 7 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.3

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Inside the **Control Options** of each task, and in the **Additional options** for a phase in a release pipeline, 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**.

> In TFS 2017.3, custom task conditions are available in the user interface only for Build pipelines. You can use the Release REST APIs to establish custom conditions for Release pipelines.

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	-

		TO				
	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 pipelines.

Concurrent release jobs in Team Foundation Server

7/19/2018 • 4 minutes 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 newer. We don't charge you for Team Foundation Build (TFBuild) so long as you have a TFS Client Access License (CAL).

A TFS *concurrent job* gives you the ability to run a single release at a time in a project collection. You can keep hundreds or even thousands of release jobs in your collection. But, to run more than one release at a time, you need additional concurrent jobs.

One free concurrent job is included with every collection in a Team Foundation server. Every Visual Studio Enterprise subscriber in a Team Foundation server contributes one additional concurrent job. You can buy additional private jobs from the Visual Studio Marketplace.

> Do I need concurrent jobs in TFS 2015? Short answer: no. More details

How a concurrent job is consumed

For example, a collection in a Team Foundation server has one concurrent job. This allows users in that collection to run only one release at a time. When additional releases are triggered, they are queued and will wait for the previous one to complete.

A release requires a concurrent job only when it is being actively deployed to an environment. Waiting for an approval does not consume a concurrent job. However, waiting for a manual intervention in the middle of a deployment does consume a concurrent job.

1. FabrikamFiber Release 10 is first to be deployed.
2. Deployment of FabrikamFiber Release 11 starts after Release 10's deployment is complete.
3. Release 12 is queued until Release 11's deployment is active.
4. Release 11 waits for an approval. Release 12's deployment starts because a release waiting for approvals does not consume a concurrent job.
5. Even though Release 11 is approved, it resumes only after Release 12's deployment is completed.
6. Release 11 is waiting for manual intervention. Release 13 cannot start because the manual intervention state consumes a concurrent job.

> Manual intervention does not consume a job in TFS 2017.1 and newer.

Concurrent processing within a single release

Concurrent processing within a single release does not require additional concurrent jobs. 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 jobs. You can have more than three agents running at the same time to perform parallel operations within releases. For instance, notice below that four or five agents are actively running jobs from three concurrent jobs.

Concurrent jobs in an organization

For example, here's an organization that has multiple Team Foundation Servers. Two of their users have Visual Studio Enterprise subscriptions that they can use at the same time across all their on-premises servers and in each collection so long as the customer adds them as users to both the servers as explained below.

Determine how many concurrent jobs you need

You can begin by seeing if your teams can get by with the concurrent jobs you've got by default. As the number of queued releases exceeds the number of concurrent jobs you have, your release queues will grow longer. When you find the queue delays are too long, you can purchase additional concurrent jobs as needed.

Simple estimate

A simple rule of thumb: Estimate that you'll need one concurrent job for every 10 users in your server.

Detailed estimate

In the following scenarios you might need multiple concurrent jobs:

- If you have multiple teams, if each of them require a CI build, and if each of the CI builds is configured to trigger a release, then you'll likely need a concurrent job for each team.
- If you develop multiple applications in one collection, then you'll likely need additional concurrent jobs: one to deploy each application at the same time.

Use your Visual Studio Enterprise subscription benefit

Users who have Visual Studio Enterprise subscriptions are assigned to **VS Enterprise** access level in the Users hub of TFS instance. Each of these users contributes one additional concurrent job to each collection. You can use this benefit on all Team Foundation Servers in your organization.

1. Browse to **Server settings**, **Access levels**.

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 jobs

If you need to run more concurrent releases, you can buy additional private jobs from the Visual Studio marketplace. Since there is no way to directly purchase concurrent jobs from Marketplace for a TFS instance at present, you must first buy concurrent jobs for a VSTS organization. After you buy the private jobs for a VSTS organization, you enter the number of purchased concurrent jobs manually on the resource limits page described below.

View and manage concurrent jobs

1. Browse to **Collection settings**, **Build and Release**, **Resource limits**.

URL example: `http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue?_a=resourceLimits`

2. View or edit the number of purchased concurrent jobs.

Q&A

Who can use the system?

TFS users with a TFS CAL can author as many releases as they want.

To approve releases, a TFS CAL is not necessary; any user with stakeholder access can approve or reject releases.

Do I need concurrent jobs to run builds on TFS?

No, on TFS you don't need concurrent jobs to run builds. You can run as many builds as you want at the same time for no additional charge.

Do I need concurrent jobs 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.

How is release management licensed in VSTS?

See concurrent jobs in VSTS.

CI/CD Concurrent Jobs in Team Services

7/19/2018 • 6 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
>
> May 2018: For the past few months we have not enforced throttling of concurrent builds or releases on self-hosted agents because of an issue with our design. As a result, your ability to run multiple builds or releases concurrently on these agents was limited only by the number of agents in your organization.
>
> We've fixed the issue, and beginning in June 2018 we'll resume throttling and you'll be able to run only as many concurrent build or release jobs as explained below.

A CI/CD *concurrent job* gives you the ability to run a single build job or a single release job at a time in your organization. In VSTS you can run concurrent jobs on Microsoft-hosted infrastructure or on your own (self-hosted) infrastructure.

Microsoft-hosted CI/CD

formerly hosted pipelines

If you want to run builds and releases on machines managed by Microsoft, then you need Microsoft-hosted CI/CD. We provide a **free tier** of service by default in your VSTS organization, allowing you to run one concurrent job for up to 240 minutes per month. This uses our pool of Microsoft-hosted agents to run your builds and releases. Each build or release job can run for up to 30 minutes.

When the **free tier** of Microsoft-hosted CI/CD is no longer sufficient, you can pay for CI/CD capacity per concurrent job. Paid Microsoft-hosted CI/CD removes the monthly time limit and also allows you to run a single job for up to 6 hours.

Buy Microsoft-hosted CI/CD

Self-hosted CI/CD

formerly private pipelines

If you want to run builds and releases on your own machines, then you need *self-hosted CI/CD*. You'll start by deploying our self-hosted agents on your machines. You can register any number of these self-hosted agents in your VSTS organization for free. We charge based on the number of jobs you want to run at a time, not the number of agents registered.

Your VSTS organization includes one self-hosted concurrent job free to get you started. Additionally, for each active Visual Studio Enterprise subscriber who is a member of your organization you get one additional self-hosted concurrent job. If you need additional capacity, you can buy more. There are no monthly time limits on self-hosted CI/CD.

Buy self-hosted CI/CD

How a concurrent job is consumed by a build or release

For example, consider a VSTS organization that has only one Microsoft-hosted concurrent job. This allows users

in that organization to collectively run only one build or release job at a time. When additional jobs are triggered, they are queued and will wait for the previous job to complete.

A release consumes a concurrent job only when it is being actively deploying to an environment. While the release is waiting for an approval or a manual intervention, it does not consume a concurrent job.

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 job.
5. Release 11 is approved. It resumes only after Fabrikam CI Build 101 is completed.

How a concurrent job is consumed by a phase

A single build or release can comprise of multiple jobs through phases and environments.

- An agent phase can create multiple jobs through multi-configuration or multi-agent options. Each of these consumes a concurrent job. If there are not enough concurrent jobs, then the jobs are queued up and run one after the other.
- A server phase does not create any jobs, and hence does not consume any concurrent jobs.
- A deployment group phase creates as many jobs as there are machines in the deployment group. Jobs that run on deployment group machines do not consume any concurrent jobs.

Determine how many concurrent jobs you need

You can begin by seeing if your teams can get by with the free tier of Microsoft-hosted CI/CD and self-hosted CI/CD offered in your VSTS organization. When you've reached the 240 minute limit for the free tier of Microsoft-hosted CI/CD, then you can start by buying one concurrent job to remove this monthly time limit before deciding to increase the number of concurrent jobs further.

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

Simple estimate

A simple rule of thumb: Estimate that you'll need one concurrent job for every 4-5 users in your organization.

Detailed estimate

In the following scenarios you might need multiple concurrent jobs:

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

View available concurrent jobs

1. Browse to **Organization settings**, **Build and Release**, **Resource limits**.

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

2. View the maximum number of concurrent jobs of Microsoft-hosted and self-hosted CI/CD that are available in your organization.
3. Select **Pipelines queue...** to display all the builds and releases that are actively consuming an available concurrent job or that are queued waiting for a concurrent job to be available.

Sharing of concurrent jobs across projects in a collection

Concurrent jobs are purchased at the organization level, and they are shared by all projects in an organization. We don't yet offer a way to partition or dedicate concurrent job capacity to a specific project or agent pool. For example:

1. You purchase two concurrent jobs in your organization.
2. You queue two builds in the first project, and both the concurrent jobs 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 concurrent jobs.

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 pipelines that I can create?

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

I use XAML build controllers with my organization. How am I charged for those?

You can register one XAML build controller for each self-hosted concurrent job in your organization. Your

organization gets at least one free self-hosted concurrent job, so you can register one XAML build controller for no additional charge. For each additional XAML build controller, you'll need an additional self-hosted concurrent job.

As a Visual Studio Enterprise subscriber, do I get additional concurrent jobs for TFS and VSTS?

Yes. Visual Studio Enterprise subscribers get one concurrent job in Team Foundation Server 2017 or later and one self-hosted concurrent job in each VSTS organization of which they are a member.

Library

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Library is a collection of *shared* build and release assets for a project. Assets defined in a library can be used in multiple build and release pipelines of the project. The **Library** tab can be accessed directly in 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 pipelines in VSTS and TFS 2017 and newer at present. Task groups and service connections are available to build and release pipelines in TFS 2015 and newer, and 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 pipelines. For example, you must be a 'User' for a variable group to be able to use it in a release pipeline.
Administrator	In addition to all the above operations, members of this role can manage membership of all other roles for the item. The user that created an item is automatically added to the Administrator role for that item.

The security settings for the **Library** tab control access for *all* items in the library. Role memberships for individual items are automatically inherited from those of the **Library** node. In addition to the three roles listed above, the **Creator** role on the library defines who can create new items in the library, but cannot be used to manage permissions for other users. By default, the following groups are added to the **Administrator** role of the library: **Build Administrators**, **Release Administrators**, and **Project Administrators**.

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Variable groups for builds and releases

7/10/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Use a variable group to store values that you want to make available across multiple build and release pipelines. Variable groups are defined and managed in the **Library** tab of the **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**.

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**.

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.

2. Specify your Azure subscription end point and the name of the vault containing your secrets.

   Ensure the Azure service connection has at least **Get** and **List** management permissions on the vault for secrets. You can enable 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 pipeline linked to the variable group during the build or release.
- Any changes made to *existing* secrets in the key vault, such as a change in the value of a secret, will be made available automatically to all the definitions in which the variable group is used.
- When *new* secrets are added to the vault, 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 your build or release pipeline, select the **Variables** tab, select **Variable groups**, and then choose **Link variable group**. In a build pipeline, you see a list of available groups. In a release pipeline (as shown below), you also see a drop-down list of environments in the pipeline - you can link the variable group to one or more of these environments.

- In a **build pipeline**, the variable group is linked to the pipeline and all the variables in the group are available for use within this pipeline.
- In a **release pipeline**, you can link a variable group to the pipeline itself, or to a specific environment of the release pipeline.
  - If you link to a release pipeline, all the variables in the group are available for use in the pipeline and in

all environments of that pipeline.

- If you link to one or more environments in a release pipeline, the variables from the variable group are scoped to these environments and are not accessible in the other environments of the same release.

> **NOTE**
> Linking a variable group to a specific environment is available only on VSTS and on TFS 2018 Update 2 and later.

You access the value of the variables in a linked variable group in exactly the same way as variables you define within the pipeline itself. For example, to access the value of a variable named **customer** in a variable group linked to the pipeline, use `$(customer)` in a task parameter or a script. However, secret variables (encrypted variables and key vault variables) cannot be accessed directly in scripts - instead they must be passed as arguments to a task.

> **Note:** At present, variables in different groups that are linked to a pipeline in the same scope (such as a release or environment scope) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

Any changes made centrally to a variable group, such as a change in the value of a variable or the addition of new variables, will automatically be made available to all the definitions or 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 for builds and releases

7/18/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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

The new task group is automatically added to the task catalogue, ready to be added to other release and build pipelines. Task groups are stored at the project level, and are not accessible outside the project scope.

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

Before you create a task group...

- Ensure that all of the tasks you want to include in a task group have their parameters defined as variables, such as **$(MyVariable)**, where you want to be able to configure these parameters when you use the task group. Variables used in the tasks are automatically extracted and converted into parameters for the task group. Values of these configuration variables will be converted into default values for the task group.
- If you specify a value (instead of a variable) for a parameter, that value becomes a fixed parameter value and cannot be exposed as a parameter to the task group.
- Parameters of the encapsulated tasks for which you specified a value (instead of a variable), or you didn't provide a value for, are not configurable in the task group when added to a build or release pipeline.
- Task conditions (such as "Run this task only when a previous task has failed" for a **PowerShell Script** task) can be configured in a task group and these settings are persisted with the task group.
- When you save the task group, you can provide a name and a description for the new task group, and select a category where you want it to appear in the **Task catalog** dialog. You can also change the default values for each of the parameters.
- When you queue a build or a release, the encapsulated tasks are extracted and the values you entered for the task group parameters are applied to the tasks.
- Changes you make to a task group are reflected in every instance of the task group.

Create a task group

1. Select a sequence of tasks in a build or release pipeline (when using a mouse, click on the checkmarks of each one). Then open the shortcut menu and choose **Create task group**.

2. Specify a name and description for the new task group, and the category (tab in the Add tasks panel) you want to add it to.

3. After you choose **Create**, the new task group is created and replaces the selected tasks in your pipeline.

4. Save your updated pipeline.

Manage task groups

All the task groups you create in the current project are listed in the **Task Groups** tab of the **Build & Release** hub.

Use the **Export** shortcut command to save a copy of the task group as a JSON pipeline, and the **Import** icon to import previously saved task group definitions. Use this feature to transfer task groups between projects and enterprises, or replicate and save copies of your task groups.

Select a task group name to open the details page.

- In the **Tasks** page you can edit the tasks that make up the task group. For each encapsulated task you can change the parameter values for the non-variable parameters, edit the existing parameter variables, or convert parameter values to and from variables. When you save the changes, all definitions that use this task group will pick up the changes.
- In the **History** tab you can see the history of changes to the group.
- In the **References** tab you can expand lists of all the build and release pipelines, and other task groups, that use (reference) this task group. This is useful to ensure changes do not have unexpected effects on other processes.

Create previews and updated versions of task groups

All of the built-in tasks in VSTS and TFS are versioned. This allows build and release pipelines to continue to use the existing version of a task while new versions are developed, tested, and released. In VSTS, you can version your own custom task groups so that they behave in the same way and provide the same advantages.

1. After you finish editing a task group, choose **Save as draft** instead of **Save**.

2. The string **-test** is appended to the task group version number. When you are happy with the changes, choose **Publish draft**. You can choose whether to publish it as a preview or as a production-ready version.

3. You can now use the updated task group in your build and release processes; either by changing the version number of the task group in an existing pipeline or by adding it from the **Add tasks** panel.

As with the built-in tasks, the default when you add a task group is the highest non-preview version.

4. After you have finished testing the updated task group, choose **Publish preview**. The **Preview** string is removed from the version number string. It will now appear in definitions as a "production-ready" version.

5. In a build or release pipeline that already contains this task group, you can now select the new "production-ready" version. When you add the task group from the **Add tasks** panel, it automatically selects the new "production-ready" version.

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 connections for builds and releases

7/19/2018 • 20 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You will typically need to connect to external and remote services to execute tasks for a build or deployment. For example, you may need to connect to your Microsoft Azure subscription, to a different build server or file server, to an online continuous integration environment, or to services you install on remote computers.

You can define service connections in Visual Studio Team Services (VSTS) or Team Foundation Server (TFS) that are available for use in all your tasks. For example, you can create a service connection for your Azure subscription and use this service connection name in an Azure Web Site Deployment task in a release pipeline.

You define and manage service connections from the Admin settings of your project.

- VSTS: `https://{organization}.visualstudio.com/{teamproject}/_admin/_services`
- TFS: `https://{tfsserver}/{collection}/{teamproject}/_admin/_services`

Service connections are created at project scope. A service connection created in one project is not visible in another project.

Create a service connection

1. Open the **Services** page from the "settings" icon in the top menu bar.

2. Choose **+ New service connection** and select the type of service connection you need.

3. Fill in the parameters for the service connection. The list of parameters differs for each type of service connection - see the following list. For example, this is the default **Azure Resource Manager** connection dialog:

4. Choose **OK** to create the connection.

> You can also create your own custom service connections.

Secure a service connection

You can control who can define new service connections in a library, and who can use an existing service connection. **Roles** are defined for service connections, and **membership** in these roles governs the operations you can perform on those service connections.

ROLE ON A LIBRARY SERVICE CONNECTION	PURPOSE
User	Members of this role can use the service connection when authoring build or release pipelines.

ROLE ON A LIBRARY SERVICE CONNECTION	PURPOSE
Administrator	In addition to using the service connection, members of this role can manage membership of all other roles for the service connection. The user that created the service connection is automatically added to the Administrator role for that service connection.

Two special groups called **Service connection administrators** and **Service connection creators** are added to every project. Members of the Service connection administrators group can manage all service connections. By default, project administrators are added as members of this group. This group is also added as an administrator to every service connection created. Members of the Service connection creators group can create new service connections. By default, project contributors are added as members of this group.

To modify the security for a connection:

1. Open the **Services** page from the "settings" icon in the top menu bar.

2. Choose the **Roles** link to open the security tab.

3. Add users or groups, turn on and off inheritance, or change the role for existing users and groups as required.

Use a service connection

After the new service connection is created:

- If you are using it in the UI, select the connection name you assigned in the **Azure subscription** (or the

equivalent connection name) setting of your pipeline.

- If you are using it in YAML, copy the connection name into your code as the **azureSubscription** (or the equivalent connection name) value.

> You can also create your own custom service connections.

Common service connection types

VSTS and TFS support a variety of service connection types by default. Some of these are described below:

- Azure Classic service connection
- Azure Resource Manager service connection
- Azure Service Bus service connection
- Bitbucket Cloud service enconnectiondpoint
- Chef service connection
- Docker Host service connection
- Docker Registry service connection
- External Git service connection
- Generic service connection
- GitHub service connection
- GitHub Enterprise service connection
- Jenkins service connection
- Kubernetes service connection
- npm service connection

- NuGet service connection
- Service Fabric service connection
- SSH service connection
- Subversion service connection
- Team Foundation Server / VSTS service connection
- Visual Studio App Center service connection

After you enter the parameters when creating a service connection, validate the connection. The validation link uses a REST call to the external service with the information you entered, and indicates if the call succeeded.

Azure Classic service connection

Defines and secures a connection to a Microsoft Azure subscription using Azure credentials or an Azure management certificate. How do I create a new service connection?

PARAMETER	DESCRIPTION
[authentication type]	Required. Select **Credentials** or **Certificate based**.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Environment	Required. Select **Azure Cloud**, **Azure Stack**, or one of the pre-defined Azure Government Clouds where your subscription is defined.
Subscription ID	Required. The GUID-like identifier for your Azure subscription (not the subscription name). You can copy this from the Azure portal.
Subscription Name	Required. The name of your Microsoft Azure subscription (account).
User name	Required for Credentials authentication. User name of a work or school account (for example @fabrikam.com). Microsoft accounts (for example @live or @hotmail) are not supported.
Password	Required for Credentials authentication. Password for the user specified above.
Management Certificate	Required for Certificate based authentication. Copy the value of the management certificate key from your publish settings XML file or the Azure portal.

> If your subscription is defined in an Azure Government Cloud, ensure your application meets the relevant compliance requirements before you configure a service connection.

Azure Resource Manager service connection

Defines and secures a connection to a Microsoft Azure subscription using Service Principal Authentication (SPA). The dialog offers two modes:

- **Automated subscription detection**. In this mode, VSTS and TFS will attempt to query Azure for all of

the subscriptions and instances to which you have access using the credentials you are currently logged on with in VSTS or TFS (including Microsoft accounts and School or Work accounts). If no subscriptions are shown, or subscriptions other than the one you want to use, you must sign out of VSTS or TFS and sign in again using the appropriate account credentials.

- **Manual subscription pipeline**. In this mode, you must specify the service principal you want to use to connect to Azure. The service principal specifies the resources and the access levels that will be available over the connection. Use this approach when you need to connect to an Azure account using different credentials from those you are currently logged on with in VSTS or TFS. This is also a useful way to maximize security and limit access.

For more information, see Create an Azure service connection

NOTE: If you don't see any Azure subscriptions or instances, or you have problems validating the connection, see Troubleshoot Azure Resource Manager service connections.

Azure Service Bus service connection

Defines and secures a connection to a Microsoft Azure Service Bus queue.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Service Bus ConnectionString	The URL of your Azure Service Bus instance. More information.
Service Bus Queue Name	The name of an existing Azure Service Bus queue.

How do I create a new service connection?

Bitbucket service connection

Defines a connection to a Bitbucket server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

How do I create a new service connection?

Chef service connection

Defines and secures a connection to a Chef automation server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Chef automation server.
Node Name (Username)	Required. The name of the node to connect to. Typically this is your username.
Client Key	Required. The key specified in the Chef .pem file.

How do I create a new service connection?

Docker Host service connection

Defines and secures a connection to a Docker host.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Docker host.
CA Certificate	Required. A trusted certificate authority certificate to use to authenticate with the host.
Certificate	Required. A client certificate to use to authenticate with the host.
Key	Required. The key specified in the Docker key.pem file.

Ensure you protect your connection to the Docker host. Learn more.

How do I create a new service connection?

Docker Registry service connection

Defines and secures a connection to a Docker registry.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Docker Registry	Required. The URL of the Docker registry. A default value is provided.

PARAMETER	DESCRIPTION
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.

How do I create a new service connection?

External Git service connection

Defines and secures a connection to a Git repository server. Note that there is a specific service connection for GitHub and GitHub Enterprise connections.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Git repository server.
User name	Required. The username to connect to the Git repository server.
Password/Token Key	Required. The password or access token for the specified username.

Also see Artifact sources.

How do I create a new service connection?

Generic service connection

Defines and secures a connection to any other type of service or application.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
User name	Required. The username to connect to the service.
Password/Token Key	Required. The password or access token for the specified username.

How do I create a new service connection?

GitHub service connection

Defines a connection to a GitHub repository. Note that there is a specific service connection for External Git servers and GitHub Enterprise connections.

PARAMETER	DESCRIPTION
Choose authorization	Required. Either **Grant authorization** or **Personal access token**. See notes below.
Token	Required for Personal access token authorization. See notes below.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.

How do I create a new service connection?

> **NOTE**
>
> If you select **Grant authorization** for the **Choose authorization** option, the dialog shows an **Authorize** button that opens the GitHub login page. If you select **Personal access token** you must obtain a suitable token and paste it into the **Token** textbox. The dialog shows the recommended scopes for the token: **repo, user, admin:repo_hook**. See this page on GitHub for information about obtaining an access token. Then register your GitHub account in your profile:

- Open your profile from your organization name at the right of the 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.

GitHub Enterprise service connection

Defines a connection to a GitHub repository. Note that there is a specific service connection for External Git servers and standard GitHub service connections.

PARAMETER	DESCRIPTION
Choose authorization	Required. Either **Personal access token** or **Username and Password**. See notes below.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
Accept untrusted SSL certificates	Set this option to allow clients to accept a self-signed certificate instead of installing the certificate in the TFS service role or the computers hosting the agent.

PARAMETER	DESCRIPTION
Token	Required for Personal access token authorization. See notes below.
User name	Required for Username and Password authentication. The username to connect to the service.
Password	Required for Username and Password authentication. The password for the specified username.

How do I create a new service connection?

> **NOTE**
> If you select **Personal access token** you must obtain a suitable token and paste it into the **Token** textbox. The dialog shows the recommended scopes for the token: **repo, user, admin:repo_hook**. See this page on GitHub for information about obtaining an access token. Then register your GitHub account in your profile:

- Open your profile from your account name at the right of the 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.

Jenkins service connection

Defines a connection to the Jenkins service.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the service.
Accept untrusted SSL certificates	Set this option to allow clients to accept a self-signed certificate instead of installing the certificate in the TFS service role or the computers hosting the agent.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

How do I create a new service connection?

Also see VSTS Integration with Jenkins and Artifact sources.

Kubernetes service connection

Defines and secures a connection to a Kubernetes automation account.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server URL	Required. The URL of the Kubernetes automation service.
Kubeconfig	The contents of the kubectl configuration file.

How do I create a new service connection?

npm service connection

Defines and secures a connection to an npm server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Registry URL	Required. The URL of the npm server.
Username	Required when connection type is **Basic authentication**. The username for authentication.
Password	Required when connection type is **Basic authentication**. The password for the username.
Personal Access Token	Required when connection type is **External VSTS**. The token to use to authenticate with the service. Learn more.

How do I create a new service connection?

NuGet service connection

Defines and secures a connection to a NuGet server.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Feed URL	Required. The URL of the NuGet server.
ApiKey	Required when connection type is **ApiKey**. The authentication key.

PARAMETER	DESCRIPTION
Personal Access Token	Required when connection type is **External VSTS**. The token to use to authenticate with the service. Learn more.
Username	Required when connection type is **Basic authentication**. The username for authentication.
Password	Required when connection type is **Basic authentication**. The password for the username.

How do I create a new service connection?

Service Fabric service connection

Defines and secures a connection to a Service Fabric cluster.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Cluster Endpoint	Required. The TCP endpoint of the cluster.
Server Certificate Thumbprint	Required when connection type is **Certificate based** or **Azure Active Directory**.
Client Certificate	Required when connection type is **Certificate based**.
Password	Required when connection type is **Certificate based**. The certificate password.
Username	Required when connection type is **Azure Active Directory**. The username for authentication.
Password	Required when connection type is **Azure Active Directory**. The password for the username.
Use Windows security	Required when connection type is **Others**.
Cluster SPN	Required when connection type is **Others** and usiong Windows security.

How do I create a new service connection?

SSH service connection

Defines and secures a connection to a remote host using Secure Shell (SSH).

PARAMETER	DESCRIPTION

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Host name	Required. The name of the remote host machine or the IP address.
Port number	Required. The port number of the remote host machine to which you want to connect. The default is port 22.
User name	Required. The username to use when connecting to the remote host machine.
Password or passphrase	The password or passphrase for the specified username if using a keypair as credentials.
Private key	The entire contents of the private key file if using this type of authentication.

How do I create a new service connection?

Also see SSH task and Copy Files Over SSH.

Subversion service connection

Defines and secures a connection to the Subversion repository.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Server repository URL	Required. The URL of the repository.
Accept untrusted SSL certificates	Set this option to allow the client to accept self-signed certificates installed on the agent computer(s).
Realm name	Optional. If you use multiple credentials in a build or release pipeline, use this parameter to specify the realm containing the credentials specified for this service connection.
User name	Required. The username to connect to the service.
Password	Required. The password for the specified username.

How do I create a new service connection?

Team Foundation Server / VSTS service connection

Defines and secures a connection to another TFS or VSTS organization.

PARAMETER	DESCRIPTION
(authentication)	Select **Basic** or **Token Based** authentication.
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
Connection URL	Required. The URL of the TFS or 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. Learn more.

How do I create a new service connection?

Use the **Verify connection** link to validate your connection information.

See also Authenticate access with personal access tokens for VSTS and TFS.

Visual Studio App Center service connection

Defines and secures a connection to Visual Studio App Center.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure account or subscription. If you are using YAML, use this name as the **azureSubscription** or the equivalent subscription name value in the script.
API Token	Required. The token to use to authenticate with the service. Learn more.

How do I create a new service connection?

Extensions for other service connections

Other service connection types and tasks can be installed in VSTS and Team Foundation Server as extensions. Some examples of service connections currently available through extensions are:

- TFS artifacts for Release Management. Deploy on-premises TFS builds with VSTS Release Management through a TFS service connection 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 connection, 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.

> You can also create your own custom service connections.

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

7/17/2018 • 2 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Use the **Secure Files** library to store files such as signing certificates, Apple Provisioning Profiles, Android Keystore files, and SSH keys on the server without having to commit them to your source repository. Secure files are defined and managed in the **Library** tab 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 pipelines in the project based on the security settings. Secure files follow the library security model.

There's a size limit of 10 MB for each secure file.

Q & A

How can I consume secure files in a Build or Release Pipeline?

Use the Download Secure File Utility task to consume secure files within a Build or Release Pipeline.

How can I create a custom task using secure files?

You can build your own tasks that use secure files by using inputs with type `secureFile` in the `task.json`. Learn how to build a custom task.

The Install Apple Provisioning Profile task is a simple example of a task using a secure file. See the reference documentation and source code.

To handle secure files during build or release, you can refer to the common module available here.

My task can't access the secure files. What do I do?

Make sure your agent is running version of 2.116.0 or higher. See Agent version and upgrades.

Build and release retention policies

7/19/2018 • 8 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Retention policies are used to configure how long builds and releases are to be retained by the system. The primary reasons to delete older builds and releases are to conserve storage and to reduce clutter. The main reasons to keep builds and releases are for audit and tracking.

Build retention

In most cases you don't need to retain completed builds longer than a certain number of days. Using build retention policies, you can control **how many days** you want to keep each build before deleting it and the **minimum number of builds** that should be retained for each pipeline.

As an author of a build pipeline, you can customize retention policies for builds of your pipeline on the **Retention** tab. You can also customize these policies on a branch-by-branch basis if you are building from Git repositories.

Global build retention policy

If you are using an on-premises Team Foundation Server, you can specify build retention policy defaults and maximums for a project collection. You can also specify when builds are permanently destroyed (removed from the **Deleted** tab in the build explorer).

If you are using VSTS, you can view but not change these settings for your organization.

Global build retention policy settings can be managed from the **Build and Release** settings of your organization or project collection:

- VSTS: `https://{your_organization}.visualstudio.com/_admin/_buildQueue`
- TFS 2017 and newer: `https://{your_server}/tfs/DefaultCollection/_admin/_buildQueue`
- TFS 2015.3: `http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue`
- TFS 2015 RTM: `http://{your_server}:8080/tfs/DefaultCollection/_admin/_buildQueue#_a=settings`

The **maximum retention policy** sets the upper limit for how longs builds can be retained for all build pipelines. Authors of build pipelines cannot configure settings for their definitions beyond the values specified here.

The **default retention policy** sets the default retention values for all the build pipelines. Authors of build pipelines can override these values.

The **build destruction policy** helps you keep the builds for a certain period of time after they are deleted. This policy cannot be overridden in individual build pipelines.

Git repositories

If your repository type is one of the following, you can define multiple retention policies with branch filters:

- 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 pipeline meets the above requirements:

When specifying custom policies for each pipeline, you cannot exceed the maximum limits set by administrator.

Clean up pull request builds

If you protect your Git branches with pull request builds, then you can use retention policies to automatically delete the completed builds. To do it, add a policy that keeps a minimum of `0` builds with the following branch filter:

```
refs/pull/*
```

TFVC and Subversion repositories

For TFVC and Subversion repository types you can modify a single policy with the same options shown above.

Policy order

When the system is purging old builds, it evaluates each build against the policies in the order you have specified. You can drag and drop a policy lower or higher in the list to change this order.

The "All" branches policy is automatically added as the last policy in the evaluation order to enforce the maximum limits for all other branches.

What parts of the build get deleted

When the retention policies mark a build for deletion, you can control which information related to the build is deleted:

- Build record: You can choose to delete the entire build record or keep basic information about the build even after the build is deleted.
- Source label: If you label sources as part of the build, then you can choose to delete the tag (for Git) or the label (for TFVC) created by a build.
- Automated test results: You can choose to delete the automated test results associated with the build (for example, results published by the Publish Test Results build task).

The following information is deleted when a build is deleted:

- Logs
- Published artifacts
- Published symbols

When are builds deleted

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 pipeline determine how long a release and the build linked to it are retained. Using these policies, you can control **how many days** you want to keep each release after it has been last modified or deployed and the **minimum number of releases** that should be retained for each pipeline. The retention timer on a release is reset every time a release is modified or deployed to an environment. The minimum number or releases to retain setting takes precedence over the number of days. For example, if you specify to retain a minimum of three releases, the most recent three will be retained indefinitely - irrespective of the number of days specified. However, you can manually delete these releases when you no longer require them.

As an author of a release pipeline, you can customize retention policies for releases of your pipeline on the **Retention** tab. You can also customize these policies on 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 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 project.

Global release retention policy settings can be managed from the **Release** settings of your project:

- VSTS:
  ```
  https://{your_organization}.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 pipelines. Authors of release pipelines cannot configure settings for their definitions beyond the values specified here.

The **default retention policy** sets the default retention values for all the release pipelines. Authors of build pipelines can override these values.

The **destruction policy** helps you keep the releases for a certain period of time after they are deleted. This policy cannot be overridden in individual release pipelines.

> In TFS, release retention management is restricted to specifying the number of days, and this is available only in TFS 2015.3 and newer.

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 pipeline meets the above requirements:

In this example, if a release that is deployed to Dev is not promoted to QA for 10 days, it is a potential candidate for deletion. However, if that same release is deployed to QA eight days after being deployed to Dev, its retention timer is reset, and it is retained in the system for another 30 days.

When specifying custom policies per pipeline, you cannot exceed the maximum limits set by administrator.

Interaction between build and release retention

The build linked to a release has its own retention policy, which may be shorter than that of the release. If you want to retain the build for the same period as the release, set the **Retain build** checkbox for the appropriate 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 pipeline, delete a release, or when the retention policy deletes a release automatically, the retention policy for the associated build will determine when that build is deleted.

> In TFS, interaction between build and release retention is available in TFS 2017 and newer.

Q&A

Are manual test results deleted?

No

If I mark a build or a release to be retained indefinitely, does the retention policy still apply?

No. Neither the pipeline's retention policy nor the maximum limits set by the administrator are applied when you mark an individual build or release to be retained indefinitely. It will remain until you stop retaining it indefinitely.

How do I specify that builds deployed to production will be retained longer?

Customize the retention policy on the release pipeline. Specify the number of days that releases deployed to production must be retained. In addition, indicate that builds associated with that release are to be retained. This will override the build retention policy.

I did not mark builds to be retained indefinitely. However, I see a large number of builds being retained. How can I prevent this?

Builds that are deployed as part of releases are also governed by the release retention policy. Customize the release retention policy as explained above.

Are automated test results that are published as part of a release retained until the release is deleted?

Test results published within 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 or VSTS Build through Release Management, and are still publishing test results, the retention of these results is governed by the retention of the release they belong to.

Create an Azure service connection

7/17/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This topic explains how to create an Azure Resource Manager service connection for connecting to Microsoft Azure resources. It starts by showing the simple case where you select the subscription, and optionally the Azure Resource Group, to which you want to connect. Use this approach:

- If you are connecting from VSTS, and not from TFS.
- If you are the owner of both the Azure and the VSTS subscriptions you are connecting from, and both accept the same credentials as you are currently signed into VSTS with.
- You do not need to further limit the permissions for Azure resources accessed through the service connection.
- You are not connecting to Azure Stack or an Azure Government Cloud.

If you have problems using this simple approach (such as no subscriptions being shown in the drop-down list), or if you want to further limit users' permissions, you can do so by using a service principal as shown here.

Create an Azure Resource Manager service connection

1. Open the **Services** page from the "settings" icon in the top menu bar.

2. Choose **+ New service connection** and select **Azure Resource Manager**.

3. Fill in the following parameters for the service connection.

PARAMETER	DESCRIPTION
Connection Name	Required. The name you will use to refer to this service connection in task properties. This is not the name of your Azure subscription.
Subscription	Select an existing Azure subscription. If you don't see any Azure subscriptions or instances, see Troubleshoot Azure Resource Manager service connections.
Resource Group	Leave empty to allow users to access all resources defined within the subscription - users will be able to access only the resources defined within that group. Or select a resource group to which you want to restrict the users' access - users will be able to access only the resources defined within that group.

4. After the new service connection is created:

   - If you are using it in the UI, select the connection name you assigned in the **Azure subscription** setting of your pipeline.
   - If you are using it in YAML, copy the connection name into your code as the **azureSubscription** value.

See also: Troubleshoot Azure Resource Manager service connection.

Create an Azure Resource Manager service connection with an existing service principal

1. If you want to use a pre-defined set of access permissions, and you don't already have a suitable service principal defined, follow one of these tutorials to create a new service principal:

   - Use the portal to create an Azure Active Directory application and service principal that can access

resources

- How to create and test Azure Service Principal using Azure CLI

2. Open the **Services** page from the "settings" icon in the top menu bar.

3. Choose **+ New service connection** and select **Azure Resource Manager**.

4. Switch from the simplified version of the dialog to the full version using the link in the dialog.

5. Enter a user-friendly name to use when referring to this service connection.
6. Select the Environment name (such as Azure Cloud, Azure Stack, or an Azure Government Cloud).
7. Enter the Environment URL if required. For Azure Stack, this will be something like `https://management.local.azurestack.external`
8. Download and run this PowerShell script in an Azure PowerShell window. When prompted, enter your subscription name, password, role (optional), and the type of cloud such as Azure Cloud (the default), Azure Stack, or an Azure Government Cloud.
9. 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
10. After the new service connection is created:
    - If you are using it in the UI, select the connection name you assigned in the **Azure subscription** setting of your pipeline.
    - If you are using it in YAML, copy the connection name into your code as the **azureSubscription** value.
11. If required, modify the service principal to expose the appropriate permissions. For more details, see Use Role-Based Access Control to manage access to your Azure subscription resources. This blog post also contains more information about using service principal authentication.

See also: Troubleshoot Azure Resource Manager service connections.

Connect to an Azure Government Cloud

For information about connecting to an Azure Government Cloud, see:

- Connecting from Visual Studio Team Services (Azure Government Cloud)

Connect to Azure Stack

For information about connecting to Azure Stack, see:

- Connect to Azure Stack
- Connect Azure Stack to Azure using VPN
- Connect Azure Stack to Azure using ExpressRoute

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

How to use YAML builds

7/19/2018 • 3 minutes to read • Edit Online

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 pipelines give you the advantages of configuration as code.

> **NOTE**
> To use YAML you must have the **Build YAML pipelines** preview feature enabled on your organization.

In a YAML build pipeline, your CI build process configured as code, which means:

- The pipeline 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 Build a repo with YAML file.

How do YAML builds compare to web-interface builds?

CAPABILITY	YAML	WEB INTERFACE
Repository		
Bitbucket	Not yet	Yes
Git and GitHub	Yes	Yes
TFVC	No	Yes
External Git, Subversion	No	Yes
Capabilities		
Allow scripts to access OAuth token	Yes	Yes
Badge enabled	Not yet	Yes
Build in a container	Yes	Not yet
Build job authorization scope	Not yet	Yes
Checkout options: Tag sources, Report build status, Checkout submodules	Not yet	Yes

CAPABILITY	YAML	WEB INTERFACE
Configuration as code	Yes	No
Create work item on failure	Not yet	Yes
Fan out and fan in	Yes	Yes
Link work items	Not yet	Yes
Pause/disable pipeline from web editor	Not yet	Yes
Set environment variables for scripts (useful for secrets)	Yes	Not yet
Task groups	No	Yes
Variables across phases	Yes	Not yet

Automatically create a YAML build pipeline

To make it more convenient to create YAML build pipelines, VSTS automatically creates a pipeline when you add a file named .vsts-ci.yml to the root of your repository. It creates the build pipeline 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 project already has a build pipeline that's pointing to the file, then the system does not automatically create another build pipeline.

Manually create a YAML build pipeline

If your code is in GitHub, or if you want to create multiple YAML build pipelines, then after you have published the YAML file in your repo, you can manually create a build pipeline.

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 pipeline.
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 Microsoft-hosted agent pools, including **Hosted VS 2017**, **Hosted Linux** or **Hosted macOS**. You'll get different kinds of greetings from each of these typs 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/CopyFilesV2 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

7/19/2018 • 2 minutes 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 pipeline

1. Open your project in your web browser ▼

   (If you don't see your project listed on the home page, select **Browse**.)

   - On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_team_project}`
   - VSTS: `https://{your_organization}.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 pipeline (Build and Release tab > Builds) ▼

3. Select the **Azure Cloud Service** template.
4. As the repository source, select the project, repository, and branch.
5. Remove the **Azure Cloud Service Deployment task** from the build pipeline, since you will be deploying the cloud service later through a release pipeline.

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 pipeline 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 pipeline will consume to deploy your app.

Deploy your app

After you've run the build, you're ready to create a release pipeline 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 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.

Build your SQL server database

7/19/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 pipeline

1. Open your project in your web browser ▼

(If you don't see your project listed on the home page, select **Browse**.)

- On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_team_project}`
- VSTS: `https://{your_organization}.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 pipeline (Build and Release tab > Builds) ▼

3. Select the **.NET Desktop** template.
4. As the repository source, select the 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 pipeline 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 pipeline will consume to deploy your database.

Deploy your database

After you've run the build, you're ready to create a release pipeline 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 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

7/17/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

When you are ready to move beyond the basics of compiling and testing your code, use a PowerShell script to add your team's business logic to your build process.

You can run a PowerShell Script on a Windows build agent.

1. Push your script into your repo.
2. Add a PowerShell build task.

3. Drag the build task where you want it to run.
4. Specify the name of the script.

Example: Version your assemblies

For example, to version to your assemblies, copy and upload this script to your project:

```
##-----------------------------------------------------------------------
## <copyright file="ApplyVersionToAssemblies.ps1">(c) Microsoft Corporation. This source is subject to the
Microsoft Permissive License. See
http://www.microsoft.com/resources/sharedsource/licensingbasics/sharedsourcelicenses.mspx. All other rights
reserved.</copyright>
##-----------------------------------------------------------------------
# Look for a 0.0.0.0 pattern in the build number.
# If found use it to version the assemblies.
#
# For example, if the 'Build number format' build 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 $Env:BUILD_BUILDNUMBER))
{
    Write-Error "You must set the following environment variables"
    Write-Error "to test this script interactively."
    Write-Host '$Env:BUILD_SOURCESDIRECTORY - For example, enter something like:'
    Write-Host '$Env:BUILD_SOURCESDIRECTORY = "C:\code\FabrikamTFVC\HelloWorld"'
    Write-Host '$Env:BUILD_BUILDNUMBER - For example, enter something like:'
    Write-Host '$Env:BUILD_BUILDNUMBER = "Build HelloWorld_0000.00.00.0"'
    exit 1
}

# Make sure path to source code directory is available
if (-not $Env:BUILD_SOURCESDIRECTORY)
{
    Write-Error ("BUILD_SOURCESDIRECTORY environment variable is missing.")
    exit 1
}
elseif (-not (Test-Path $Env:BUILD_SOURCESDIRECTORY))
{
    Write-Error "BUILD_SOURCESDIRECTORY does not exist: $Env:BUILD_SOURCESDIRECTORY"
    exit 1
}
Write-Verbose "BUILD_SOURCESDIRECTORY: $Env:BUILD_SOURCESDIRECTORY"

# Make sure there is a build number
if (-not $Env:BUILD_BUILDNUMBER)
{
    Write-Error ("BUILD_BUILDNUMBER environment variable is missing.")
    exit 1
}
Write-Verbose "BUILD_BUILDNUMBER: $Env:BUILD_BUILDNUMBER"

# Get and validate the version data
$VersionData = [regex]::matches($Env:BUILD_BUILDNUMBER,$VersionRegex)
switch($VersionData.Count)
{
   0
      {
         Write-Error "Could not find version number data in BUILD_BUILDNUMBER."
         exit 1
      }
   1 {}
   default
      {
         Write-Warning "Found more than instance of version data in BUILD_BUILDNUMBER."
```

```
            Write-Warning "Will assume first instance is version."
        }
    }
    $NewVersion = $VersionData[0]
    Write-Verbose "Version: $NewVersion"

    # Apply the version to the assembly property files
    $files = gci $Env:BUILD_SOURCESDIRECTORY -recurse -include "*Properties*","My Project" |
        ?{ $_.PSIsContainer } |
        foreach { gci -Path $_.FullName -Recurse -include AssemblyInfo.* }
    if($files)
    {
        Write-Verbose "Will apply $NewVersion to $($files.count) files."

        foreach ($file in $files) {
            $filecontent = Get-Content($file)
            attrib $file -r
            $filecontent -replace $VersionRegex, $NewVersion | Out-File $file
            Write-Verbose "$file.FullName - version applied"
        }
    }
    else
    {
        Write-Warning "Found no files."
    }
```

Add the build task to your build pipeline.

Specify your build number with something like this:

```
$(BuildDefinitionName)_$(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 pipeline and select **Allow Scripts to Access OAuth Token**.

After you've done that, your script can use to SYSTEM_ACCESSTOKEN 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"
$pipeline = Invoke-RestMethod -Uri $url -Headers @{
    Authorization = "Bearer $env:SYSTEM_ACCESSTOKEN"
}
Write-Host "Pipeline = $($pipeline | ConvertTo-Json -Depth 1000)"
```

Q&A

What variables are available for me to use in my scripts?

Use variables

How do I set a variable so that it can be read by subsequent scripts and tasks?

Define and modify your build variables in a script

Define and modify your release variables in a script

Which branch of the script does the build run?

The build runs the script same branch of the code you are building.

What kinds of parameters can I use?

You can use named parameters. Other kinds of parameters, such as switch parameters, are not yet supported and will cause errors.

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

For some workflows you need your build 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 Microsoft-hosted agents and on on-premises agents.

Enable scripts to run Git commands

Grant version control permissions to the build service

Go to the Version Control control panel tab ▼

- VSTS: `https://{your-organization}.visualstudio.com/DefaultCollection/{your-team-project}/_admin/_versioncontrol`
- On-premises: `https://{your-server}:8080/tfs/DefaultCollection/{your-team-project}/_admin/_versioncontrol`

If you see this page, select the repo, and then click the link:

On the **Version Control** tab, select the repository in which you want to run Git commands, and then select **Project Collection Build Service**.

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 pipeline 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 task:

TASK	ARGUMENTS
Utility: Command Line List the files in the Git repo.	**Tool**: `git` **Arguments**: `ls-files`

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 task:

TASK	ARGUMENTS
Utility: Batch Script Run merge.bat.	**Path**: `merge.bat`

Q & A

Can I run Git commands if my remote repo is in GitHub or an external Git service such as Bitbucket?

Yes

Which tasks can I use to run Git commands?

Batch Script

Command Line

PowerShell

Shell Script

How do I avoid triggering a CI build when the script pushes?

Add `***NO_CI***` to your commit message. For example,

```
git merge origin/features/hello-world -m "Merge to master ***NO_CI***"
```

How does enabling scripts to run Git commands affect how the build process gets build sources?

When you set `system.prefergit` 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

7/9/2018 • 8 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 pipeline from the Android or Xamarin.Android build template. Or, if you already have a build pipeline, add the Android Signing task after the task that builds your APK.
3. Find the Android Signing task's **Sign the APK** checkbox and enable it.
4. Next to the **Keystore file** field, click the settings icon and upload your keystore file to the Secure Files library. During upload, your keystore will be encrypted and securely stored.
5. Once your keystore has been uploaded to the Secure Files library, select it in the **Keystore file** dropdown.
6. Go to the **Variables** tab and add the following variables. In their **Value** column, enter your **Keystore password**, **Key alias**, and **Key password**.
   - **keystore-password**: Password to the unencrypted keystore file. *Be sure to click the **lock** icon.* This will secure your password and obscure it in logs.
   - **key-alias**: The key alias for the signing certificate you generated.
   - **key-password**: The password for the key associated with the specified alias. *Again, be sure to click the **lock** icon.*

7. Go back to the **Tasks** tab and reference the names of your newly-created variables in the signing options.

Save your build pipeline, and you are all set! Any build agent will now be able to securely sign your app without any certificate management on the build machine itself.

Sign your Apple iOS, macOS, tvOS, or watchOS app

For your Xcode or Xamarin.iOS build to sign and provision your app, it needs access to your P12 signing certificate and one or more provisioning profiles. The following sections explain how to obtain these files.

Obtain your P12 signing certificate

After creating your development or distribution signing certificate, export it to a `.p12` file using either Xcode or the Keychain Access app on macOS.

1. To export using Xcode 8 or lower, go to **Xcode** > **Preferences...** > **Accounts** and select your Apple Developer account.
2. Click **View Details...**, right-click on the signing identity you wish to export, and click **Export...**.
3. Enter a filename and password. Take note of the password as you will need it later.

4. Alternatively, follow a similar process using the **Keychain Access** app on macOS or generate a signing certificate on Windows. Use the procedure described in this article if you prefer this method.

Obtain your provisioning profile

You can download your app provisioning profile from the Apple Developer portal, unless your app uses automatic signing. You can also use Xcode to access those that are installed on your Mac.

1. Using Xcode 8 or lower, go to **Xcode** > **Preferences...** > **Accounts** and select your Apple Developer account.
2. Right-click the provisioning profile you want to use and select **Show in Finder**.
3. Copy the highlighted file from Finder to another location and give it a descriptive filename.

Configure your build

There are two recommended ways for your build to access signing certificates and provisioning profiles for signing and provisioning your app:

1. Installing them during the build
2. Preinstalling them on a macOS build agent

Choose either of the tabs below for details.

- Install them during the build
- Preinstall them on a macOS build agent

Use this method when you do not have enduring access to the build agent, such as the hosted macOS agents. The P12 certificate and provisioning profile are installed at the beginning of the build and removed when the build completes.

Install the P12 certificate during your build

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 pipeline, 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.

How To: Extend your deployments to IIS Deployment Groups

7/9/2018 • 2 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

You can quickly and easily deploy your ASP.NET or Node 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 pipeline, build artifacts, and websites required.

Dynamically create and remove a deployment group

You can create and remove deployment groups dynamically if you prefer by using the Azure Resource Group Deployment task to install the agent on the machines in a deployment group using ARM templates. See Provision deployment group agents.

Apply 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 pipeline; for example, `<connectionStringKeyName> = <value>`.
2. In the **IIS Web App Deploy** task, select the checkbox for **XML variable substitution** under **File Transforms and Variable Substitution Options**.

   > If you prefer to manage 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**.

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 pipeline, 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 Web Deploy package to IIS servers using WinRM

7/17/2018 • 7 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

> A simpler way to deploy web applications to IIS servers is by using deployment groups instead of WinRM. Deployment groups are currently in preview for some users in 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

JOINED TO A	PROTOCOL	ADDRESSING MODE
Domain	HTTP	FQDN

Ensure that your IIS servers are set up in one of these configurations. For example, do not use WinRM over HTTP to communicate with a Workgroup machine. Similarly, do not use an IP address to access the target server(s) when you use HTTP. Instead, in both scenarios, use HTTPS.

> If you need to deploy to a server that is not in the same workgroup or domain, add it to trusted hosts in your WinRM configuration.

Follow these steps to configure each target server.

1. Enable File and Printer Sharing. You can do this by executing the following command in a Command window with Administrative permissions:

   ```
   netsh advfirewall firewall set rule group="File and Printer Sharing" new enable=yes
   ```

2. Check your PowerShell version. You need PowerShell version 4.0 or above installed on every target machine. To display the current PowerShell version, execute the following command in the PowerShell console:

   ```
   $PSVersionTable.PSVersion
   ```

3. Check your .NET Framework version. You need version 4.5 or higher installed on every target machine. See How to: Determine Which .NET Framework Versions Are Installed.

4. Download from GitHub this PowerShell script for Windows 10 and Windows Server 2016, or this PowerShell script for previous versions of Windows. Copy them to every target machine. You will use them to configure WinRM in the following steps.

5. Decide if you want to use HTTP or HTTPS to communicate with the target machine(s).

   - If you choose HTTP, execute the following in a Command window with Administrative permissions:

     ```
     ConfigureWinRM.ps1 {FQDN} http
     ```

     > This command creates an HTTP WinRM listener and opens port 5985 inbound for WinRM over HTTP.

   - If you choose HTTPS, you can use either a FQDN or an IP address to access the target machine(s). To use a FQDN to access the target machine(s), execute the following in the PowerShell console with Administrative permissions:

     ```
     ConfigureWinRM.ps1 {FQDN} https
     ```

     To use an IP address to access the target machine(s), execute the following in the PowerShell console with Administrative permissions:

     ```
     ConfigureWinRM.ps1 {ipaddress} https
     ```

     > These commands create a test certificate by using **MakeCert.exe**, use the certificate to create an HTTPS WinRM listener, and open port 5986 inbound for WinRM over HTTPS. The script also increases the WinRM **MaxEnvelopeSizekb** setting. By default on Windows Server this is 500 KB, which can result in a "Request size exceeded the configured MaxEnvelopeSize quota" error.

IIS configuration

If you are deploying an ASP.NET app, make sure that you have ASP.NET 4.5 or ASP.NET 4.6 installed on each of your IIS target servers. For more information, see this topic.

If you are deploying an ASP.NET Core application to IIS target servers, follow the additional instructions in this topic to install .NET Core Windows Server Hosting Bundle.

If you are deploying a Node application to IIS target servers, follow the instructions in this topic to install and configure IISnode on IIS servers.

In this example, we will deploy to the Default Web Site on each of the servers. If you need to deploy to another website, make sure you configure this as well.

IIS WinRM extension

Install the IIS Web App Deployment Using WinRM extension from Visual Studio Marketplace in VSTS 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 pipeline that's automatically linked to the build pipeline.
   - Open the **Releases** tab of the **Build & Release** hub, open the **+** drop-down in the list of release pipelines, and choose **Create release pipeline**.
2. Choose **Start with an Empty process**.
3. If you created your new release pipeline from a build summary, check that the build pipeline and artifact is shown in the **Artifacts** section on the **Pipeline** tab. If you created a new release pipeline from the **Releases** tab, choose the **+ Add** link and select your build artifact.

4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.

5. On the **Variables** tab of the environment in release pipeline, 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.

IIS Deploy: WinRM - IIS Web App Deployment - Deploy the package.

- **Machines**: `$(WebServers)`
- **Admin Login**: Enter the administrator credentials for target servers. For workgroup-joined computers, use the format `.\username`. For domain-joined computers, use the format `domain\username`.
- **Password**: Enter the administrator password for target servers.
- **Protocol**: Select `HTTP` or `HTTPS` (depending on how you configured the target machine earlier). Note that if the target machine is workgroup-joined, you must choose `HTTPS`. You can use HTTP only if the target machine is domain-joined and configured to use a FDQN.
- **Web Deploy Package**: Fully qualified path of the zip file you copied to the target server in the previous task.
- **Website Name**: `Default Web Site` (or the name of the website if you configured a different one earlier).

7. Edit the name of the release pipeline, click **Save**, and click **OK**. Note that the default 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 pipeline with the artifacts produced by a specific build. This will result in deploying the build to IIS servers:

1. Choose **+ Release** and select **Create Release**.
2. Check that the build you just completed is selected 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 app and verify its contents.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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 apps to Azure Government Cloud

7/10/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2017 | TFS 2018

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Azure Government Clouds provide private and semi-isolated locations for specific Government or other services, separate from the normal Azure services. Highest levels of privacy have been adopted for these clouds, including restricted data access policies.

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 an Azure Resource Manager service connection. You must use the full version of the service connection dialog to manually define the connection. Before you configure a service connection, you should also ensure you meet all relevant compliance requirements for your application.

You can then use the service connection in your build and release pipeline tasks.

Next

- Deploy an Azure web app
- Troubleshoot Azure Resource Manager service connections

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Run unit tests with your builds

7/16/2018 • 2 minutes to read • Edit Online

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

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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

This example shows how to run unit tests with your build for .NET and ASP.NET apps. It uses the Visual Studio Test task.

> Typically you will run unit tests in your build workflow, and functional tests in your release workflow after your app is deployed (usually to a QA environment). Code coverage is available only in the build workflow.

Before you start

- Check in your solution to VSTS. Include your test projects.

Create a build pipeline

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

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

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 Visual Studio Test task version 2 supports Test Impact Analysis. For information about all the task settings, see Visual Studio Test task.

How do I pass parameters to my test code from a build pipeline?

3. If you also want to test code coverage, set the **Code coverage enabled** checkbox in the **Execution options** section.

When tests are run with this option, code coverage information is collected dynamically and assemblies do not need to be instrumented. By default, all assemblies are profiled for collecting coverage information. If you need to exclude specific assemblies and customize code coverage, use a .runsettings file.

How do I collect and publish code coverage data if I'm not using the Visual Studio Test task?

4. When you're done, save your build pipeline.

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.

Next step

Review your test results

Get started with Selenium testing in a CD pipeline

7/17/2018 • 5 minutes to read • Edit Online

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

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Performing user interface (UI) testing as part of the release process is a great way of detecting unexpected changes, and need not be difficult. This topic describes using Selenium to test your website during a continuous deployment release and test automation.

> Typically you will run unit tests in your build workflow, and functional (UI) tests in your release workflow after your app is deployed (usually to a QA environment).

For more information about Selenium browser automation, see:

- Selenium HQ
- Selenium documentation

Create your test project

As there is no template for Selenium testing, the easiest way to get started is to use the Unit Test template. This automatically adds the test framework references and enables you run and view the results from Visual Studio Test Explorer.

1. In Visual Studio, open the **File** menu and choose **New Project**, then choose **Test** and select **Unit Test Project**. Alternatively, open the shortcut menu for the solution and choose **Add** then **New Project** and then **Unit Test Project**.

2. After the project is created, add the Selenium and browser driver references used by the browser to execute the tests. Open the shortcut menu for the Unit Test project and choose **Manage NuGet Packages**. Add the following packages to your project:

   - Selenium.WebDriver
   - Selenium.Firefox.WebDriver
   - Selenium.WebDriver.ChromeDriver
   - Selenium.WebDriver.IEDriver

3. Create your tests. For example, the following code creates a default class named **MySeleniumTests** that performs a simple test on the Bing.com website. Replace the contents of the **TheBingSearchTest** function with the Selenium code required to test your web app or website. Change the **browser** assignment in the **SetupTest** function to the browser you want to use for the test.

```
using System;
using System.Text;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using OpenQA.Selenium;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.IE;

namespace SeleniumBingTests
{
  /// <summary>
  /// Summary description for MySeleniumTests
  /// </summary>
  [TestClass]
  public class MySeleniumTests
  {
    private TestContext testContextInstance;
    private IWebDriver driver;
    private string appURL;

    public MySeleniumTests()
    {
    }

    [TestMethod]
    [TestCategory("Chrome")]
    public void TheBingSearchTest()
    {
      driver.Navigate().GoToUrl(appURL + "/");
      driver.FindElement(By.Id("sb_form_q")).SendKeys("VSTS");
      driver.FindElement(By.Id("sb_form_go")).Click();
      driver.FindElement(By.XPath("//ol[@id='b_results']/li/h2/a/strong[3]")).Click();
      Assert.IsTrue(driver.Title.Contains("VSTS"), "Verified title of the page");
    }

    /// <summary>
    ///Gets or sets the test context which provides
    ///information about and functionality for the current test run.
    ///</summary>
    public TestContext TestContext
```

```
    {
      get
      {
        return testContextInstance;
      }
      set
      {
        testContextInstance = value;
      }
    }

    [TestInitialize()]
    public void SetupTest()
    {
      appURL = "http://www.bing.com/";

      string browser = "Chrome";
      switch(browser)
      {
        case "Chrome":
          driver = new ChromeDriver();
          break;
        case "Firefox":
          driver = new FirefoxDriver();
          break;
        case "IE":
          driver = new InternetExplorerDriver();
          break;
        default:
          driver = new ChromeDriver();
          break;
      }

    }

    [TestCleanup()]
    public void MyTestCleanup()
    {
      driver.Quit();
    }
  }
}
```

4. Run the Selenium test locally using Test Explorer and check that it works.

Define your CI build process

You'll need a continuous integration (CI) build process that builds your Selenium tests. For more details, see Build your .NET desktop app for Windows.

Create your web app

You'll need a web app to test. You can use an existing app, or deploy one in your CD release process. The example code above runs tests against Bing.com. For details of how to set up your own release process to deploy a web app, see Deploy to Azure web apps.

Decide how you will deploy and test your app

You can deploy and test your app using either the Microsoft-hosted agent in Azure, or a self-hosted agent that you install on the target servers.

- When using the **Microsoft-hosted agent**, you should use the Selenium web drivers that are pre-installed on the Microsoft-hosted agents because they are compatible with the browser versions installed on the

Microsoft-hosted agent images. The file paths to these drivers can be obtained from the environment variables named `IEWebDriver` (Internet Explorer), `ChromeWebDriver` (Google Chrome), and `GeckoWebDriver` (Firefox). For example,

```
driver = new ChromeDriver(Environment.GetEnvironmentVariable("ChromeWebDriver"));
```

- When using a **self-hosted agent** that you deploy on your target servers, agents must be configured to run interactively with auto-logon enabled. See Build and Release Agents.

Include the test in a CD release

NOTE: This example uses the **Visual Studio Test Platform Installer** task and the latest version of the **Visual Studio Test** task. These tasks are not available in TFS 2015 or TFS 2017. To run Selenium tests in these versions of TFS, you must use the Visual Studio Test Agent Deployment and Run Functional Tests tasks instead.

1. If you don't have an existing release pipeline that deploys your web app:

   - Open the **Releases** page in the **Build & Release** hub and choose the **+** icon, then choose **Create release pipeline**.

   - Select the **Azure App Service Deployment** template and choose **Apply**.
   - In the **Artifacts** section of the **Pipeline** tab, choose **+ Add**. Select your build artifacts and choose **Add**.

- Choose the **Continuous deployment trigger** icon in the **Artifacts** section of the **Pipeline** tab. In the Continuous deployment trigger pane, enable the trigger so that a new release is created from every build.

- Open the **Tasks** tab, select the **Environment 1** section, and enter your subscription information and the name of the web app where you want to deploy the app and tests. These settings are applied to the **Deploy Azure App Service** task.

2. If you are deploying your app and tests to environments where the target machines that host the agents do not have Visual Studio installed:

- In the **Tasks** tab of the release pipeline, choose the **+** icon in the **Run on agent** section. Select the **Visual Studio Test Platform Installer** task and choose **Add**. Leave all the settings at the default values.

You can find a task more easily by using the search textbox.

3. In the **Tasks** tab of the release pipeline, choose the **+** icon in the **Run on agent** section. Select the **Visual Studio Test** task and choose **Add**.

4. If you added the **Visual Studio Test Platform Installer** task to your pipeline, change the **Test platform version** setting in the **Execution options** section of the **Visual Studio Test** task to **Installed by Tools Installer**.

How do I pass parameters to my test code from a release pipeline?

5. Save the release pipeline and start a new release. You can do this by queuing a new CI build, or by choosing **Create release** from the **Release** drop-down list in the release pipeline.

View the test results

1. To view the test results, open the release summary from the **Releases** tab.

2. In the release summary page choose the **Tests** link to show a page that highlights the changes including errors, stack traces, and the ability to easily create a bug that contains this information.

Next steps

Review your test results

Testing Java applications with VSTS

7/16/2018 • 2 minutes to read • Edit Online

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

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Visual Studio Team Services (VSTS) and Team Foundation Server (TFS) include build tasks to make it easy to integrate your build and testing efforts into your release pipelines with rich reporting highlighting build-on-build changes, easy-to-create actionable bugs for regressions and the ability to create and display code coverage with a simple checkbox.

See this topic on Channel9 video

Build integration

VSTS build tasks make it easy to build to build Java applications. These same tasks also make it easy to test your Java applications. The example below uses Maven to build a Java application, runs JUnit tests, and uses Maven and to test the application.

The output of these tests and other testing frameworks is made available in the standard test reports by using the **Publish Test Results** task. Note that, in the image below the task defaults to importing the test results from JUnit.

Build-on-build summary results

VSTS automatically adds build-on-build changes in the summary view to give a clear indication if your quality is improving. In the example below you can see 16 new tests were added, but 6 of these tests are now failing and the build time is now much longer.

Test details view

While Summary View gives build-on-build changes, Details View lets you identify the problems and easily action them. In this view, all the failed test information is displayed. Selecting a failed test displays all the data you need to take action for this issue, such as the error message, duration, failing agent, and stack trace.

Easily create actionable bugs

After an issue has been identified, Test Details view makes it easy to create actionable bugs that include all the data from Details View, as well as automatically linking the build, test run, and a generated title.

Code coverage results

By checking an option in the test build task, code coverage is automatically collected during the tests. Summary View shows a roll-up of the data and a detailed coverage results report that enables you to drill into your application and identify coverage in different areas.

See also

VSTS Java website

Help and support

Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Review continuous test results after a build

7/16/2018 • 2 minutes to read • Edit Online

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

After your build finishes running continuous tests using Visual Studio Team Services (VSTS) or Team Foundation Server (TFS), review your test results to start analyzing any problems that you found.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Review your test results

1. In VSTS or TFS, go to your project.
2. Find your build and open the build summary.

3. Review the summary for your test run results. 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.

4. If you collected code coverage information, the build **Summary** page displays an overview of the blocks and lines covered. You can download the code coverage results for a drill-down analysis in Visual Studio.

Organize your test results

1. Choose the **Show filter bar** icon.

2. Organize your test results using the group and outcome lists.

Debug failed tests

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

2. To trace the results of a test across builds, view the test history.

3. In the history page, review when a test started to pass or fail.

Group the list of results using the drop-down list. Mouse over a bar in the chart to see a pop-up summary, or select a bar to view the detailed test results for that build.

View tests in the Release hub

- Test results are also summarized in the release summaries in the **Release** hub. Choose any of the percentage values to see the full summary in the **Test** hub.

Next step

Run automated tests from test plans in the Test hub

Set up environments to run continuous test tasks with your build tasks

7/16/2018 • 14 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To test your app using different platforms and configurations using test automation, set up separate environments to run your app and tests with your builds in Visual Studio Team Services (VSTS) or Team Foundation Server (TFS).

Set up machines to run your app and tests

You'll need to set up physical or virtual machines to run your app and tests, for example:

- Windows Server 2012 R2 or higher with IIS to run your app
- Machines with the necessary browsers to run your tests

With VSTS, you can define environments that have physical and virtual machines, such as Azure VMs and Azure resource groups. With TFS, you can define environments using only physical machines. Alternatively, you can create a virtual network isolated environment for your build-deploy-test scenarios.

If you want to use a PowerShell script to deploy your app, make sure to:

- Include that script in your solution or project.
- Enable PowerShell Remote on all your machines.

You'll need to install the agent that runs your tests on the machines. For more details, see Deploy a Windows build agent. You might decide to create Azure VMs to install your agents.

Define a list of machines to run your app and tests

> **NOTE**
> Previous versions of VSTS and TFS included the capability to define **Machine Groups**. However, this feature is no longer available.

As an alternative, consider:

- If you use version 2.x or higher of the Visual Studio Test task you can deploy and run unit and functional tests without requiring the **Deploy Test Agent** and **Run Functional Tests** tasks, and run tests on platforms that don't have Visual Studio installed by using the Visual Studio Test Platform. In this case, you can use deployment groups to define your target machines. For more details, see Testing with unified agents and phases.

- A **comma-delimited list** of machine IP addresses or fully-qualified domain names (FQDNs), together with any port information, in all your build or release pipelines. For example:

  ```
  dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.12.34:5986
  ```

- A variable that contains the list of machines. For example, a build or release pipeline variable or a variable defined within a project-wide variable group. For example, you could define the variable named **MyMachines** with the value shown above, then include it in your tasks using:

  ```
  $(MyMachines)
  ```

  Using a variable means that you can change the list of machines in one place and have the change apply to all the tasks that use the variable.

  > If you don't specify a port number, the default (based on the selected protocol) will be used. If you are using HTTPS, the IP address or name of the machine should match the CN entry in the certificate. Note that you can set the **Test Certificate** option in some build, test, and deploy tasks to omit certificate checks.

Run tests in parallel

The Visual Studio Test Platform (VSTest) supports running tests in parallel. Parallel test execution is available:

- To all frameworks and within the IDE, the command line (CLI), and in VSTS.
- Within the IDE from all launch points (Test Explorer, CodeLens, various **Run** commands, and more).

Parallel test execution:

- Composes with test adapters and frameworks that already support parallel execution such as MSTest, NUnit, and xUnit.net.
- Is easy to implement, it requires no changes to existing test code and does not break existing test runs.
- Works with the test code where it already resides.
- Is OFF by default, users must explicitly opt in.
- Is supported at assembly level.

Not all existing test code might be parallel-safe; for example, tests may assume exclusive use of global resources. In general, use the following iterative approach to leverage the feature:

Partition tests in terms of a taxonomy as follows:

1. Pure unit tests (typically these can run in parallel).
2. Functional tests that can run in parallel with some modifications (for example, two tests that create or delete the same folder can be adapted to use unique folders).
3. Functional tests that cannot be modified to run in parallel (for example, two Coded UI tests performing mouse actions on the desktop, or two functional tests writing to the Bluetooth or IR port).

Gradually evolve the partitioning as follows:

1. Run the tests in parallel, see which tests fail, and classify them as above.
2. Fix tests in category 2 so that they can run in parallel.
3. Move tests in category 3 into a separate test run where parallel is OFF by default.

Parallel Test Execution is **not** supported in the following cases:

- If the test run is configured using a **.testsettings** file.
- For test code targeting Phone, Store, UWP app platforms.

Enable parallel tests in Visual Studio 2017 Update 1 and VS Test task v1.x

Configure a .runsettings file in the app source repository for Visual Studio IDE or the CLI, and in VSTS when using version 1.x of the Visual Studio Test task.

In the **.runsettings** file add entry for **MaxCpuCount**, and specify or associate the file with the test run. The value for **MaxCpuCount** has the following semantics:

- **n** where 1 <= n <= number of cores: up to **n** processes will be launched.
- **n** of any other value: The number of processes launched will be as many as the available cores on the machine.
- A value of 0 (zero) indicates that up to all the available free cores may be used.

> If you need to update an existing **.runsettings** file containing an entry for **MaxCpuCount**, see the tables in this blog post for more information.
>
> For information about in-assembly parallel test execution, see this blog post.

More information about test processes for parallel execution.

Enable parallel tests in Visual Studio 2017 Update 2 and later

Enable parallel test execution by using the button on the Test Explorer toolbar. This is an ON/OFF toggle setting.

For the CLI, **vstest.console.exe** supports a **/Parallel** command line switch. Set this to enable parallel test execution.

When parallel test execution is enabled, the value of **MaxCpuCount** is set to 0 (zero) to use all the available free cores. The value is persisted with the solution and is merged with any associated .runsettings file just before a run begins. The only way to adjust the value of **MaxCpuCount** is by explicitly adding a **.runsettings** file.

> If you need to update an existing **.runsettings** file containing an entry for **MaxCpuCount**, see the tables in this blog post for more information.
>
> For information about in-assembly parallel test execution, see this blog post.

Enable parallel tests in VSTS with VS Test task v2.x

Enable parallel test execution by setting the **Run Tests in Parallel...** checkbox in the settings for the Visual Studio Test task.

When parallel test execution is enabled, the value of **MaxCpuCount** is set to 0 (zero) to use all the available free cores. The value is persisted with the solution and is merged with any associated .runsettings file just before a run begins. The only way to adjust the value of **MaxCpuCount** is by explicitly adding a **.runsettings** file.

> If you need to update an existing **.runsettings** file containing an entry for **MaxCpuCount**, see the tables in this blog post for more information.
>
> For information about in-assembly parallel test execution, see this blog post.

Testing with unified agents and phases

Version 2 of the **Visual Studio Test** task uses the unified Build and Release agent, instead of a different custom agent as was the case with version 1. The new version of the task also integrates intelligently with task phases. This topic explores how you can use this task, and explains how it works in a range of scenarios.

For more information about the tasks see:

- Visual Studio Test version 1
- Visual Studio Test version 2

You select the specific version of a task you want to use in the **Version** list at the top of the task properties pane. Use the **i** icon to show more information about the task or a selected property setting.

Advantages of using the unified agent

- You no longer need to use dedicated machines for testing (as was required by the **Run Functional Tests** task). With the unified agent, you can leverage the common agent pool. Administrators can set up a reusable pool of machines, making management much easier.
- You can use the unified agent for single machine as well multi-machine distributed execution.
- You no longer need the **Visual Studio Test Agent Deployment** task. This task is based on WinRM, which imposes several limitations.
- You no longer need any "copy files" tasks because all execution is now local to the automation agent, and task phases download the artifacts to the target machines automatically. There is no requirement to copy test assemblies and their dependencies when running tests remotely using the **Run Functional Tests** task.

How test tasks run in phases

You can add different types of phases to a release pipeline. The properties of these phases include settings for **Parallelism**.

The following sections describe how this setting affects the operation of the **Visual Studio Test** and **Run Functional Tests** tasks. For a full description of the operation for all tasks, see Parallel execution using agent phases.

No parallelism

A single agent from the specified queue will be allocated to this phase. This is the default, and all tasks in the phase

will run on that agent. The **Visual Studio Test** task runs in exactly the same way as version 1 with single agent test execution.

For example, you could deploy an Azure web app and run a small number of quick tests on it (for which a single agent is sufficient), along with some pre- and post-test setup and cleanup activities, using an environment configured as follows:

Multiple executions

This mode is driven by 'multipliers', in much the same way as a multi-configuration Build or Release. You define the multipliers as variables. Based on the values for these variables, the various configurations are run.

In the case of Build, you typically use **BuildPlatform** and **BuildConfiguration** as multipliers. The same logic applies to testing. For example, you could deploy a web app to Azure and run cross-browser tests on IE and Firefox by configuring an environment to use two phases - one for the deploy phase and one for the test phase:

The test phase is set up as a multiple executions process using a variable named **Browser**, which has the values `IE` and `Firefox`. The phase will run twice using these two configurations - one agent is assigned the value `IE`

for its **Browser** variable, and one with the value `Firefox`.

In the tasks for the environment, the **Browser** value could be used to instantiate the appropriate browser for the tests. For example, you might pass the values as **Test Run Parameters** and access them using **TestContext** in the test code. You could also use the values to provide appropriate titles for your test runs so that, if a test fails in a particular configuration, you can easily tell which run it came from.

The execution results might look like this:

Multiple agents

Multiple agents will be allocated to the phase. You specify the number of agents to be allocated from the pool, and the set of tasks in that phase will be distributed across all these agents.

In this mode, the Visual Studio Test task runs in a special way. It recognizes that its a multiple agents phase, and runs tests in a distributed manner across all the allocated agents. Because other tasks run across all agents, any pre- and post-test tasks also run equally on all the agents. Therefore, all the agents are prepared and cleaned up in a consistent manner. In addition, test execution does not require all agents to be available at the same time. If some agents are busy with another release or build, the phase can still start with the available number of agents that match the demand, and test execution starts. As additional agents become available, they can pick up any remaining tests that have not yet run.

For example, the log from a multiple agents test run, where some tests have failed, might look like this:

Artifacts are automatically downloaded when the phase starts, so the test assemblies and other files are already located on the agent, and no "copy files" task is required. So, to publish an Azure web app and run a large number of tests with fast test execution, you could model the environment as two phases - one being the deploy phase (which runs on a single agent because you don't want multiple agents to deploy the same app concurrently), and the other a test phase that uses multiple agents mode to achieve test distribution.

This also means that you can use different agent queues for the two phases, allowing you to manage agents for different purposes separately if required.

FAQs

Q: How do I do this with Build?

A: The phases capability is currently available only in Release Management. It will become available in Build soon.

Q: Does the Visual Studio Test version 1 task behave the same way as the version 2 task?

A: No, the version 1 task cannot be used for test distribution. On the single agent (the default, no parallelism) setting, the task will run in the same way as on the previous test agent. In the multiple executions and multiple agents modes, it is replicated on the agents, in the same way as all other tasks.

Q: What is required to run UI tests?

A: To run UI tests, you must run the agent in interactive mode with auto-logon enabled. Agents set to run as a service cannot run UI tests. Also disable any screensaver and unlock the machine so that UI actions in the test are not blocked.

Q: Can I run UI tests on the Microsoft-hosted agents?

A: No, running UI tests on the Microsoft-hosted agents is not possible at present.

Q: What does the 'Test mix contains UI tests' checkbox do?

A: Currently, it is there only as a reminder to run agents interactively if you are running UI tests. If you are using an agent pool with a mix of interactive and 'running as service' agents, you may also want to add an 'Interactive' capability to your agents demand that in your test phase to ensure the appropriate set of agents that can run UI tests are used. See Build and Release agent capabilities.

Q: In multiple executions mode, do I get distribution of tests as well?

A: No, multiple executions mode assigns only one agent per configuration.

Q: How do I map the configuration in multiple executions mode to my Test Configurations using tcm.exe?

A: Currently this is not possible.

Q: How else can I use multiple executions mode?

A: This mode can be used whenever you need multiple agents to execute jobs in parallel. For more examples, see Parallel execution using agent phases.

Q: Has the Run Functional Tests task also changed?

A: No, the Run Functional Tests (RFT) task has not changed. If you are using this task you *do* need the **Deploy Test Agent** task. Note that, because tasks are replicated in the multiple agents and multiple executions mode,

using the Run Functional Tests task in this mode will lead to undesirable effects.

Q: Do I need to install Visual Studio on all the machines to use the Visual Studio Test version 2 task?

A: Currently, yes. An alternate means for running tests without requiring Visual Studio on the agent machine is under consideration, which will allow you to create a separate pool of agents for testing purposes.

Q: I am using my own test runner (not the Visual Studio Test task) in the pipeline. What happens to it?

A: In the multiple agents and multiple executions mode, the task will be replicated on each of the agents. You can use the multiple executions mode to partition your tests on different configuration using the configuration variable. For example, if you have a configuration variable named **Platform** that has values `x86` and `x64`, you can run the two sets of tests on two agents in parallel by referring to your test assemblies using `**\$(Platform)\*test*.dll`

Q: How does the Visual Studio Test version 2 task run on deployment groups?

A: Yes, the task can be used to run on deployment groups.

If you have scenarios that necessitate running tests on machines in the deployment group where the app is deployed, you can use the Visual Studio Test version 2 task. If multiple machines are selected (using, for example, tags) in a **Run on Deployment Group** phase, the tests will be replicated to each of the machines.

See Also

- MSTest V2 in-assembly parallel test execution
- Parallel Test Execution
- Create a virtual network isolated environment for build-deploy-test scenarios
- Speed up testing with Test Impact Analysis

Help and support

Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Set up continuous testing for your builds

7/16/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Find problems early after changes are checked in and built by running continuous tests using Visual Studio Team Services (VSTS) or Team Foundation Server (TFS) and test automation.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

NOTE: You can use version 2.x or higher of the Visual Studio Test task to deploy and run unit and functional tests without requiring the **Deploy Test Agent** and **Run Functional Tests** tasks, and run tests on platforms that don't have Visual Studio installed by using the Visual Studio Test Platform.

For more details, see Testing with unified agents and phases.

Before you start

- Create a build pipeline that builds your solution after each check-in, using continuous integration.
- Set up environments to run your app and tests remotely in a distributed environment after each build.

Set up app deployment for your build

1. In VSTS or TFS, go to your project.
2. Go to your build pipeline and open it for editing.

3. Choose the + icon for the process phase and add tasks to deploy your app for testing. For example, if you're testing a Visual Studio solution and want to use the **Window Machine File Copy** and **PowerShell on Target Machines** tasks:

4. Add the details to copy your app from its drop location to its test environment:

   - Source folder for your app
   - List of machines where you want to deploy your app
   - Credentials to connect to the target machines
   - Target folder where to put your app

> If you use the list of machines in more than one task, consider defining a variable that contains the list of machines. For example, a build or release pipeline variable or a variable defined within a project-wide variable group. Using a variable means that you can change the list of machines in one place and have the change apply to all the tasks that use the variable.

5. Add the details to deploy your app using PowerShell.
   - List of machines where you want to deploy your app
   - Credentials to connect to the target machines
   - Path to the PowerShell script used to deploy your app

6. Save your build pipeline.

Set up test deployment for your build

NOTE: You can use version 2.x of the Visual Studio Test task to deploy and run unit and functional tests without requiring the **Deploy Test Agent** and **Run Functional Tests** tasks, and run tests on platforms that don't have Visual Studio installed by using the Visual Studio Test Platform.

If you want to use the **Deploy Test Agent** and **Run Functional Tests** tasks:

1. In your build pipeline, add a **Window Machine File Copy** task with these details to deploy your tests:

   - Source folder for your tests
   - List of machines where you want to run your tests

- Credentials to connect to the target machines
- Target folder where to put your tests

2. Add a task with these details to deploy the test agent for running your tests:
   - List of machines where you want to deploy the test agent
   - Credentials to connect to the target machines
   - Credentials for the deployment agents, so the test agent can run in your test environment

3. Save your build pipeline.

Set up your tests to run with your build

1. In your build pipeline, add a task with these details to run your tests with the test agent:

   - List of machines where you want to run your tests
   - Folder where you put your tests

2. Save your build pipeline and queue a build to check your test run.

3. After your build is done, review your test results.

See also

- Visual Studio Test task
- Visual Studio Test Platform
- Set up environments for continuous testing with your builds
- Review continuous test results after a build
- Run tests with your builds
- Speed up testing with Test Impact Analysis
- Run tests in parallel

Help and support

Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Speed up testing by using Test Impact Analysis (TIA)

7/16/2018 • 6 minutes to read • Edit Online

Visual Studio 2015.3 and later | TFS 2017.1 and later | VSTS

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Continuous Integration (CI) is a key practice in the industry. Integrations are frequent, and verified with an automated build that runs regression tests to detect integration errors as soon as possible. However, as the codebase grows and matures, its regression test suite tends to grow as well - to the extent that running a full regression test might require hours. This slows down the frequency of integrations, and ultimately defeats the purpose of continuous integration. In order to have a CI pipeline that completes quickly, some teams defer the execution of their longer running tests to a separate stage in the pipeline. However, this only serves to further defeat continuous integration.

Instead, enable Test Impact Analysis (TIA) when using the Visual Studio Test task in a build pipeline. TIA performs incremental validation by automatic test selection. It will automatically select only the subset of tests required to validate the code being committed. For a given code commit entering the CI/CD pipeline, TIA will select and run only the relevant tests required to validate that commit. Therefore, that test run will complete more quickly, if there is a failure you will get to know about it sooner, and because it is all scoped by relevance, analysis will be faster as well.

Test Impact Analysis has:

- **A robust test selection mechanism**. It includes existing impacted tests, previously failing tests, and newly added tests.
- **Safe fallback**. For commits and scenarios that TIA cannot understand, it will fall back to running all tests. TIA is currently scoped to only managed code, and single machine topology. So, for example, if the code commit contains changes to HTML or CSS files, it cannot reason about them and will fall back to running all tests.
- **Configurable overrides**. You can run all tests at a configured periodicity.

However, be aware of the following caveats when using TIA with Visual Studio 2015:

- **Running tests in parallel**. In this case, tests will run serially.
- **Running tests with code coverage enabled**. In this case, code coverage data will not get collected.

Test Impact Analysis supported scenarios

At present, TIA is supported for:

- TFS 2017 Update 1 onwards, and VSTS
- Version 2.* of the Visual Studio Test task in the build pipeline
- Build vNext, with multiple VSTest Tasks
- VS2015 Update 3 onwards on the build agent
- Local and hosted build agents
- CI and in PR workflows
- Git, GitHub, External Git, TFVC repos (including partially mapped TFVC repositories with a workaround)
- IIS interactions (over REST, SOAP APIs), using HTTP/HTTPS protocols
- Automated Tests
- Single machine topology. Tests and app (SUT) must be running on the same machine.
- Managed code (any .NET Framework app, any .NET service)

At present, TIA is **not** supported for:

- Multi-machine topology (where the test is exercising an app deployed to a different machine)
- Data driven tests
- Test Adapter-specific parallel test execution
- .NET Core
- UWP

More information about TIA scope and applications

Enable Test Impact Analysis

TIA is supported through Version 2.* of the Visual Studio Test task. If your app is a single tier application, all you need to do is to check **Run only impacted tests** in the task UI. The Test Impact data collector is automatically configured. No additional steps are required.

If your application interacts with a service in the context of IIS, you must also configure the Test Impact data collector to run in the context of IIS by using a **.runsettings** file. Here is a sample that creates this configuration:

```
<?xml version="1.0" encoding="utf-8"?>
<RunSettings>
  <DataCollectionRunSettings>
    <DataCollectors>
      <!-- This is the TestImpact data collector.-->
      <DataCollector uri="datacollector://microsoft/TestImpact/1.0"
assemblyQualifiedName="Microsoft.VisualStudio.TraceCollector.TestImpactDataCollector,
Microsoft.VisualStudio.TraceCollector, Version=15.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
friendlyName="Test Impact">
        <Configuration>
          <!-- enable IIS data collection-->
          <InstrumentIIS>True</InstrumentIIS>
          <!-- file level data collection -->
          <ImpactLevel>file</ImpactLevel>
          <!-- any job agent related executable or any other service that the test is using needs to be
profiled. -->
          <ServicesToInstrument>
            <Name>TeamFoundationSshService</Name>
          </ServicesToInstrument>
        </Configuration>
      </DataCollector>
    </DataCollectors>
  </DataCollectionRunSettings>
</RunSettings>
```

View Test Impact Analysis outcome

TIA is integrated into existing test reporting at both the summary and details levels, including notification emails.

More information about TIA and VSTS integration

Manage Test Impact Analysis behavior

You can influence the way that tests are either included or ignored during a test run:

- **Through the VSTest task UI**. TIA can be conditioned to run all tests at a configured periodicity. Setting this option is recommended, and is the means to regulate test selection.
- **By setting a build variable**. Even after TIA has been enabled in the VSTest task, it can be disabled for a specific build by setting the variable **DisableTestImpactAnalysis** to **true**. This override will force TIA to run all tests for that build. In subsequent builds, TIA will go back to optimized test selection.

When TIA opens a commit and sees an unknown file type, it falls back to running all tests. While this is good from a safety perspective, tuning this behavior might be useful in some cases. For example:

- Set the **TI_IncludePathFilters** variable to specific paths to include only these paths in a repository for which you want TIA to apply. This is useful when teams use a shared repository. Setting this variable disables TIA for all other paths not included in the setting.
- Set the **TIA_IncludePathFilters** variable to specify file types that do not influence the outcome of tests and for which changes should be ignored. For example, to ignore changes to .csproj files set the variable to the value

!*.csproj**.

> Use the minimatch pattern when setting variables, and separate multiple items with a semicolon.

To evaluate whether TIA is selecting the appropriate tests:

- Manually validate the selection. A developer who knows how the SUT and tests are architected could manually validate the test selection using the TIA reporting capabilities.
- Run TIA selected tests and then all tests in sequence. In a build pipeline, use two test tasks - one that runs only impacted Tests (T1) and one that runs all tests (T2). If T1 passes, check that T2 passes as well. If there was a failing test in T1, check that T2 reports the same set of failures.

More information about TIA advanced configuration

Provide custom dependency mappings

TIA uses dependency maps of the following form.

```
TestMethod1
  dependency1
  dependency2
TestMethod2
  dependency1
  dependency3
```

TIA can generate such a dependencies map for managed code execution. Where such dependencies reside in **.cs** and **.vb** files, TIA can automatically watch for commits into such files and then run tests that had these source files in their list of dependencies.

You can extend the scope of TIA by explicitly providing the dependencies map as an XML file. For example, you might want to support code in other languages such as JavaScript or C++, or support the scenario where tests and product code are running on different machines. The mapping can even be approximate, and the set of tests you want to run can be specified in terms of a test case filter such as you would typically provide in the VSTest task parameters.

The XML file should be checked into your repository, typically at the root level. Then set the build variable **TIA.UserMapFile** to point to it. For example, if the file is named **TIAmap.xml**, set the variable to **$(System.DefaultWorkingDirectory)/TIAmap.xml**.

For an example of the XML file format, see TIA custom dependency mapping.

See Also

- TIA overview and VSTS integration
- TIA scope and applications
- TIA advanced configuration
- TIA custom dependency mapping

Help and support

Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

FAQs for continuous testing and test automation

7/16/2018 • 11 minutes to read • Edit Online

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

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

General topics

Go to related topic >

Q: How do I run tests from different unit test frameworks with my build?

A: First, set up your test frameworks in your development tool. For example, in Visual Studio:

1. Set up the plug-in for your test framework, if you haven't already.
2. Create a folder that has the custom binaries for your plug-in. (The plug-in package for your framework is a .vsix file. Change the extension to .zip so that you can unzip it.)
3. Add these assemblies to version control and let the build controller know where to get them.
4. In your build pipeline, provide the path to the test framework in the *Path to Custom Test Adapters** parameter:

Q: I'm having problems using xUnit with .NET Core apps. Where can I get more information?

A: See the blog post Unit Tests with .NET Core and VSTS.

Q: I am not using the Visual Studio Test task to run my unit tests. Can I still collect and publish code coverage data?

A: Yes, use the Publish Code Coverage Results task.

Q: I have multiple Publish Code Coverage Results tasks in my pipeline. Do I get a merged code coverage summary?

A: Code coverage is automatically merged for only Visual Studio coverage (.coverage) files. A merged summary is not currently available for coverage files published using multiple Publish Code Coverage Results tasks.

Q: What are the typical types of tests I can run to validate my app and deployment?

A: Testing in a continuous integration and continuous deployment (CI/CD) scenario typically consists of:

1. Run **unit tests** in the CI pipeline, as demonstrated in the example above. The **Visual Studio Test** task automatically runs tests included in the app assemblies, but there is a wide range of configuration options you can specify, such as running only specific tests. See Run Tests using Visual Studio task.

2. Run **functional tests** in the early stages of the CD pipeline. These are typically Selenium (for web apps) and Coded UI tests. To do this, add the **Deploy Test Agent** and **Run Functional Tests** tasks to your release pipeline. See Testing in Continuous Integration and Continuous Deployment Workflows.

3. Run **load tests** after the app is deployed to staging and production, after it passes all functional tests. The example shown above is just a simple test that accesses a single page in the web app to validate that

deployment succeeded and the app is running successfully. You can perform must more comprehensive load testing to validate the entire app by running cloud-based load tests and Apache JMeter load tests.

Q: Can I find a specific test run?

A: Yes, type the Run ID into the search box in the left column:

Q: Can I find specific results from a test run?

A: Yes, after you find your test run, create a query in the **Filter** tab to find the test results you want:

Q: Can I deploy to a staging slot first, and then to production?

A: Yes, you can create additional deployment slots in Azure Web Apps, and specify which slot to deploy your app to. If you do not specify a slot, the default **Production** slot is used. After you deploy, you can swap an app to a different slot using the **Azure App Service Manage** task. See Azure web apps deployment.

You can use task phases and the **Manual Intervention** task in your release pipeline to pause a deployment; for example, to examine test results after the load tests have run and before the app is swapped from staging to production.

Q: How do I pass parameters to my test code from a build or release pipeline?

A: Use a runsettings file to pass values as parameters to your test code. For example, in a release that contains several environments, you can pass the appropriate app URL to each the test tasks in each one. The runsettings file and matching parameters must be specified in the Visual Studio Test task.

Test platform version

Visual Studio 2017

Settings file

$(System.DefaultWorkingDirectory)/PartsUnlimited.CI/RunSettings/webApp.runsettings

Override test run parameters

-appURL $(appURL)

Q: Where can I find details about configuring test agents?

A: See Install and configure test agents

Q: What if I want to run debug builds of native (.cpp) unit tests on the machine with the test agent?

A: Make sure that you have debug versions of the Universal C Runtime (UCRT) on the machine with the test agent, specifically these libraries: ucrtbased.dll and vcruntime140d.dll. You can include these items with your deployment. If you're running release builds of .cpp unit tests, make sure that you have Windows Update KB2999226 on your test agent machine.

Q: Where can I learn more about integrating tests with my build?

A: Try these blog posts and videos:

- Configuring Continuous Integration and Continuous Testing with Visual Studio
- Testing in Continuous Integration and Continuous Deployment Workflows
- Integrating Testing Efforts into the DevOps Process with Build vNext and Visual Studio Release Management

---

Associating tests with test cases

Go to related topic >

Q: What are the differences if I am still using a XAML build?

A: If you are using a XAML build in VSTS or TFS, you can run tests that you have associated in a Build-Deploy-Test workflow using a Lab environment. You can also run tests using Microsoft Test Manager (MTM) and a Lab environment.

Q: What types of tests are supported?

A: These are the limitations for each type of test:

- Coded UI test, Selenium tests, and unit tests written using Version 1 of the MSTest framework **can** be associated with a test case.
- Unit tests that use the.NET Core framework **cannot** be associated with a test case.
- Tests that use other test frameworks such as Version 2 of the MSTest framework, xUnit, nUnit, Chutzpah, and others **cannot** be associated with a test case.
- Associating ordered tests and generic tests **may** work, but running these tests is not supported.

Q: Can I configure work items to open in Visual Studio?

A: Yes, if you want test work items to open inside Visual Studio instead of the default VSTS or TFS UI in your web browser, change the **Work Items | General** setting from the **Tools | Options** menu in Visual Studio.

Running automated tests from the Test hub

Go to related topic >

Q: What permissions do I need to run automated tests from the test hub?

You must be a Project Contributor, or have the following permissions:

- Create releases
- Manage releases
- Edit release environment
- Manage deployment

For more information, see Set permissions for release pipelines and Release permissions.

Q: Can I override the build or environment set at the test plan level for a specific instance of test run?

A: Yes, you can do this using the **Run with options** command. Open the shortcut menu for the test suite in the left column and choose **Run with options**.

Enter the following values in the Run with options dialog and then choose **OK**:

- **Test type and runner**: Select **Automated tests using Release environment**.
- **Build**: Select the build that has the test binaries. The test results will be associated this build.

- **Release Pipeline**: Select a pipeline from the list of release pipelines that can consume the selected build artifact.
- **Release Environment**: Select the name of the environment configured in your release pipeline.

Q: Why use release environments to run tests?

A: Release Management offers a compelling orchestration workflow to obtain test binaries as artifacts and run tests. This workflow shares the same concepts used in the scheduled testing workflow, meaning users running tests in scheduled workflow will find it easy to adapt; for example, by cloning an existing scheduled testing release pipeline.

Another major benefit is the availability of a rich set of tasks in the task catalog that enable a range of activates to be performed before and after running tests. Examples include preparing and cleaning test data, creating and cleaning configuration files, and more.

Q: How does selecting "Test run" in the Visual Studio Test task version 2 work?

A: The Test management sub-system uses the test run object to pass the list of tests selected for execution. The test task looks up the test run identifier, extracts the test execution information such as the container and test method names, runs the tests, updates the test run results, and sets the test points associated with the test results in the test run. From an auditing perspective, the Visual Studio task provides a trace from the historical releases and the test run identifiers to the tests that were submitted for on-demand test execution.

Q: Should the agent run in interactive mode or as a service?

A: If you are running UI tests such as coded UI or Selenium tests, the agent on the test machines must be running in interactive mode with auto-logon enabled, not as a service, to allow the agent to launch a web browser. If you are using a headless browser such as PhantomJS, the agent can be run as a service or in interactive mode. See Build and Release Agents, Deploy an agent on Windows, and Agent pools and queues.

Q: Where can I find detailed documentation on how to run Selenium tests?

A: See Get started with Selenium testing.

Q: What happens if I select multiple configurations for the same test?

A: Currently, the on-demand workflow is not configuration-aware. In future releases, we plan to pass configuration context to the test method and report the appropriate results.

Q: What if I need to download product binaries and test binaries from different builds? Or if I need to obtain artifacts from a source such as Jenkins?

A: The current capability is optimized for a single team build to be tested on-demand using a Release Management workflow. We will evaluate support for multi-artifact releases, including non-Team Build artifacts such as Jenkins, based on user feedback.

Q: I already have a scheduled testing release pipeline. Can I reuse the same pipeline to run test on-demand, or should I create a new pipeline as shown above?

A: We recommend you use a separate release pipeline and environment for on-demand automated testing from the **Test** hub because:

- You may not want to deploy the app every time you want to run a few on-demand tests. Scheduled testing environments are typically set up to deploy the product and then run tests.
- New releases are triggered for every on-demand run. If you have many testers executing a few on-demand test runs every day, your scheduled testing release pipeline could be overloaded with releases for these runs, making it difficult to find releases that were triggered for the pipeline that contains scheduled testing and deployment to production.
- You may want to configure the Visual Studio Test task with a Test run identifier as an input so that you can trace what triggered the release. See How does selecting "Test run (for on-demand runs)" in the Visual Studio Test task work?.

Q: Can I trigger these runs and view the results in Microsoft Test Manager?

A: No. MTM will not support running automated tests against Team Foundation builds. It only works in the web-based interface for VSTS and TFS. All new manual and automated testing product development investments will be in the web-based interface. No further development is planned for MTM. See Guidance on Microsoft Test Manager usage.

Q: I have multiple testers in my team. Can they run tests from different test suites or test plans in parallel using the same release pipeline?

A: They can use the same release pipeline to trigger multiple test runs in parallel if:

- The agent pool associated with the environment has sufficient agents to cater for parallel requests. If sufficient agents are not available, runs can still be triggered but releases will be queued for processing until agents are available.
- You have sufficient concurrent jobs to enable concurrent releases. See Concurrent jobs in VSTS or Concurrent jobs in TFS for more information.
- Testers do not run the same tests in parallel. Doing so may cause results to be overwritten depending on the order of execution.

To enable multiple different test runs to execute in parallel, set the Release Management environment trigger option for behavior when multiple releases are waiting to be deployed as follows:

- If your application supports tests running in parallel from different sources, set this option to **Allow multiple releases to be deployed at the same time**.
- If your application does not support tests running in parallel from different sources, set this option to **Allow only one active deployment at a time**.

Q: What are the typical error scenarios or issues I should look out for if my tests don't run?

A: Check and resolve issues as follows:

- The release pipeline and environment in which I want to run tests are not shown after I select the build.
  - Make sure the build pipeline that is generating the build is linked as the primary artifact in the **Artifacts**

tab of the release pipeline.

- I get an error that I don't have sufficient permission to trigger a release.
  - Configure **Create releases** and **Manage deployments** permissions for the user in the **Security** menu of the release pipeline. See Release permissions.
- I get an error that no automated tests were found.
  - Check the automation status of the selected tests. Do this in the work item for the test case, or use the **Column options** link in the **Test Plans** page of the **Test** hub to add the **Automation status** column to the list of tests. See the pre-requisites section for information about automating manual tests.
- My tests didn't execute, and I suspect the release pipeline is incorrect.
  - Use the link in the **Run summary** page to access the release instance used to run the tests, and view the release logs.
- My tests go into the error state, or remain "in-progress" even after release to the environment is triggered.
  - Check if the release environment that you selected has the correct task and version selected. You must use version 2 or higher of the **Visual Studio Test** task. Version 1 of the task, and the **Run Functional Tests** task, are not supported.

Publish symbols for debugging

7/19/2018 • 2 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

> **NOTE**
>
> A symbol server is available with 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 pipeline. 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*.

Publish symbols for NuGet packages

To publish symbols for NuGet packages, include the above task in the build pipeline that produces the NuGet packages. Then the symbols will be available to all users in the VSTS organization.

Portable PDBs

If you're using Portable PDBs, you don't need to use the **Index Sources and Publish Symbols** task. For Portable PDBs, indexing is done by the build. This is a design feature of Portable PDBs and .NET.

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

7/19/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This walkthrough will cover setting up an existing build to restore NuGet packages from Package Management feeds. It assumes that you've already:

- Set up your solution to consume packages from a Package Management feed
- Created a build for that solution
- Added the correct build service identity to your feed

To build a solution that relies on NuGet packages from Package Management feeds, add the **NuGet** task (if one is not already present).

First, click **Add build tasks...**, select the **Package** category, and add the **NuGet** task. Then drag to order the task above any build tasks that require your packages.

Next, configure these options:

- **Command:** restore
- **Path to solution, packages.config, or project.json:** The path to the file that specifies the packages you want to restore

Then, select feeds to use:

- If you've checked in a NuGet.config, select **Feeds in my NuGet.config** and select the file from your repo.
- If you're using a single VSTS/TFS feed, select the **Feed(s) I select here** option and select your feed from the dropdown.

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>
```

Restoring packages from feeds in a different organization

If your NuGet.config contains feeds in a different VSTS organization (*organization*.visualstudio.com) than the organization running the build, you'll need to set up credentials for those feeds manually.

1. Select a login account (either a service account (recommended) or a user's account) that has access to the remote feed
2. Using your browser's InPrivate mode, Incognito mode, or similar, go to the VSTS organization that contains the feed, sign in with the login account you selected in step 1, click the user profile circle in the top right, and select Security
3. Create a PAT with the **Packaging (read)** scope and keep it handy
4. In the VSTS organization that contains the build, edit the build's NuGet step and ensure you're using version 2 or greater of the task, using the version selector

5. In the **Feeds and authentication** section, Ensure you've selected the **Feeds in my NuGet.config** radio button
6. Set the path to your NuGet.config in the **Path to NuGet.config**
7. In **Credentials for feeds outside this organization/collection**, click the **+**
8. In the service connection dialog that appears, enter the feed URL (make sure it matches what's in your NuGet.config) and the PAT you created in step 3
9. Save the service connection and the build, then queue a new build

Q & A

Why can't my build restore packages?

NuGet restore can fail due to a variety of issues. One of the most common issues is the introduction of a new project in your solution that requires a target framework that isn't understood by the version of NuGet your build is using. This issue generally doesn't present itself on a developer machine because Visual Studio updates the NuGet restore mechanism at the same time it adds new project types. We're looking into similar features for 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 task to select the version of NuGet for all the subsequent tasks. You can see all available versions of NuGet on nuget.org.

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" task in the catalog (it may be in the Deprecated tasks section), insert it into your build. Otherwise, insert a "NuGet" task.
2. For your NuGet/NuGet Installer task, use the version selector under the task name to select version "0.*".

3. In the Advanced section, set the NuGet Version to "Custom" and the Path to NuGet.exe as $(Build.BinariesDirectory)\nuget.exe
4. Before your NuGet task, add a "PowerShell" task, select "Inline Script" as the Type, enter this PowerShell script as the Inline Script, and enter "4.3.0" (or any version of NuGet from this list) as the Arguments.

Our thanks to GitHub user leftler for creating the original version of the PowerShell script linked above.

Use Jenkins to restore and publish packages

7/19/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization called *MyGreatFeed*. Since you're the owner of the feed, you will automatically be able to push packages to it.
- Add the URL for the feed you just generated to the nuget.config in the root of your repo.
  - Find the `<packageSources>` section of nuget.config.
  - Just before `</packageSources>`, add a line using this template: `<add key="MyGreatFeed" value="{feed_url}" />`. Change `{feed_url}` to the URL of your feed.
  - Commit this change to your repo.

```xml
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <packageSources>
    <clear />
    <add key="vss-package-management" value="
    https://www.myget.org/F/vss-package-management/api/v2" />
    <add key="MyGreatFeed" value="
    https://fabrikam.pkgs.visualstudio.com/DefaultCollection/_apis/packaging/MyGr
    eatFeed/nuget/index.json" />
  </packageSources>
  <activePackageSource>
    <add key="All" value="(Aggregate source)" />
  </activePackageSource>
</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 pipeline in Jenkins

- Ensure you have the correct plugins installed in Jenkins.
- This will be a Freestyle project. Call it "Fabrikam.Walkthrough".

- 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.

- Under Build (see screenshot below), follow these steps:
  - Choose **Execute Windows batch command**. In the **Command** box, type `init.cmd`.
  - Choose **Build a Visual Studio project or solution using MSBuild**. This task should point to msbuild.exe and FabrikamLibrary.sln.
  - Choose **Execute Windows batch command** again, but this time, use this command: `.tools\VSS.NuGet\nuget pack FabrikamLibrary\FabrikamLibrary.csproj`.

- Save this build pipeline and queue a build.
- The build's Workspace will now contain a .nupkg just like the one you built locally earlier.

Publish a package using Jenkins

These are the last walkthrough steps to publish the package to a feed:

- Edit the build pipeline in Jenkins.
- After the last build task (which runs `nuget pack`), add a new **Execute a Windows batch command** build task.
- In the new **Command** box, add these two lines:
  - The first line puts credentials where NuGet can find them:
    ```
    .tools\VSS.NuGet\nuget sources update -Name "MyGreatFeed" -UserName "%FEEDUSER%" -Password "%FEEDPASS%"
    ```
  - The second line pushes your package using the credentials saved above:
    ```
    .tools\VSS.NuGet\nuget push *.nupkg -Name "MyGreatFeed" -ApiKey VSS
    ```

Execute Windows batch command

Command

```
.tools\VSS.NuGet\nuget sources update -Name "MyGreatFeed" -UserName "%FEEDUSER%"
-Password "%FEEDPASS%"
.tools\VSS.NuGet\nuget push *.nupkg -Name "MyGreatFeed" -ApiKey VSS
```

See the list of available environment variables

Delete

- Queue another build. This time, the build machine will authenticate to VSTS and push the package to the feed you selected.

Set up Team Build and Maven

7/9/2018 • 2 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This guide covers the basics of using 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 pipeline and select the **Maven** template.
2. Fill in the path to your `POM.xml`, configure the Maven goal you'd like for your build. Authentication to your VSTS feed should happen automatically.

Migrate from XAML builds to new builds

7/19/2018 • 13 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | XAML builds

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

We introduced XAML build automation capabilities based on the Windows Workflow Foundation in Team Foundation Server (TFS) 2010. We released another version of XAML builds in TFS 2013.

After that we sought to expand beyond .NET and Windows and add support for other kinds of apps that are based on operating systems such as macOS and Linux. It became clear that we needed to switch to a more open, flexible, web-based foundation for our build automation engine. In early 2015 in 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 pipeline into a new build pipeline. The migration process is to manually create the new build pipelines that replicate what your XAML builds do.

If you're building standard .NET applications, you probably used our default templates as provided out-of-the-box. In this case the process should be reasonably easy.

If you have customized your XAML templates or added custom tasks, then you'll need to also take other steps including writing scripts, installing extensions, or creating custom tasks.

Overview of the migration effort

Here are the steps to migrate from XAML builds to newer builds:

1. If you're using a private TFS server, set up agents to run your builds.
2. To get familiar with the new build system, create a "Hello world" build pipeline.
3. Create a new build pipeline intended to replace one of your XAML build pipelines.
   a. Create a new build pipeline.
   b. Port your XAML settings.
4. On the General tab, disable the XAML build pipeline.
5. Repeat the previous two steps for each of your XAML build pipelines.
6. Take advantage of new build features and learn more about the kinds of apps you can build.
7. Learn how to customize, and if necessary extend your system.
8. When you no longer need the history and artifacts from your XAML builds, delete the XAML builds, and then the XAML build pipelines.

> **WARNING**
> After you delete the XAML builds and pipelines, you cannot get them back.

Create new build pipelines

If you're building a standard .NET app, you're probably using one of the out-of-the-box build templates such as TfvcTemplate.12.xaml or GitTemplate.12.xaml. In this case, it will probably just take you a few clicks to create build pipelines in the new build system.

1. Open your project in your web browser ▼

   (If you don't see your project listed on the home page, select **Browse**.)

   - On-premises TFS: `http://{your_server}:8080/tfs/DefaultCollection/{your_team_project}`
   - VSTS: `https://{your_organization}.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 pipeline (Build and Release tab > Builds) ▼

3. Select a template to add commonly used tasks to your build pipeline.
4. Make any necessary changes to your build pipeline to replicate your XAML build pipeline. The tasks added by the template should simply work in many cases. But if you changed process parameters or other settings in your XAML build pipelines, below are some pointers to get you started replicating those changes.

Port your XAML settings

In each of the following sections we show the XAML user interface, and then provide a pointer to the place where you can port the setting into your new build pipeline.

General tab

XAML SETTING	TFS 2017 EQUIVALENT	VSTS AND TFS 2018 AND NEWER EQUIVALENT
Build pipeline name	You can change it whenever you save the pipeline.	When editing the pipeline: On the **Tasks** tab, in left pane click **Process**, and the **Name** field appears in right pane. In the **Builds** hub (**Mine** or **All pipelines** tab), open the action menu and choose **Rename**.
Description (optional)	Not supported.	Not supported.
Queue processing	Not yet supported. As a partial alternative, disable the triggers.	Not yet supported. As an alternative, disable the triggers.

Source Settings tab

TFVC

XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Source Settings tab	On the **Repository** tab specify your mappings with Active paths as **Map** and Cloaked paths as **Cloak**.	On the **Tasks** tab, in left pane click **Get sources**. Specify your workspace mappings with Active paths as **Map** and Cloaked paths as **Cloak**.

The new build pipeline offers you some new options. The specific extra options you'll see depend on the version you're using of TFS or VSTS. If you're using VSTS, first make sure to display **Advanced settings**. See Build pipeline source repositories.

Git

XAML SETTING	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Source Settings tab	On the **Repository** tab specify the repository and default branch.	On the **Tasks** tab, in left pane click **Get sources**. Specify the repository and default branch.

The new build pipeline offers you some new options. The specific extra options you'll see depend on the version you're using of TFS or VSTS. If you're using VSTS, first make sure to display **Advanced settings**. See Build pipeline source repositories.

Trigger tab

XAML SETTING	TFS 2017 AND NEWER, VSTS EQUIVALENT
Trigger tab	On the **Triggers** tab, select the trigger you want to use: CI, scheduled, or gated.

The new build pipeline offers you some new options. For example:

- You can potentially create fewer build pipelines to replace a larger number of XAML build pipelines. This is because you can use a single new build pipeline with multiple triggers. And if you're using 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 pipeline triggers

We don't yet support the **Build even if nothing has changed since the previous build** option.

Build Defaults tab

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Build controller	On the **General** tab, select the default agent queue.	On the **Options** tab, select the default agent queue.
Staging location	On the **Tasks** tab, specify arguments to the Copy Files and Publish Build Artifacts tasks. See Build artifacts.	On the **Tasks** tab, specify arguments to the Copy Files and Publish Build Artifacts tasks. See Build artifacts.

The new build pipeline offers you some new options. For example:

- You don't need a controller, and the new agents are easier to set up and maintain. See Build and release agents.
- You can exactly specify which sets of files you want to publish as build artifacts. See Build artifacts.

Process tab

TF Version Control

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Clean workspace	On the **Repository** tab, open the **Clean** menu, and then select **true**.	On the **Tasks** tab, in left pane click **Get sources**. Display **Advanced settings**, and then select **Clean**. (We plan to change move this option out of advanced settings.)
Get version	You can't specify a changeset in the build pipeline, but you can specify one when you manually queue a build.	You can't specify a changeset in the build pipeline, but you can specify one when you manually queue a build.
Label Sources	On the **Repository** tab, select an option from the **Label sources** menu.	**Tasks** tab, in left pane click **Get sources**. Select one of the **Tag sources** options. (We plan to change the name of this to **Label sources**.)

The new build pipeline offers you some new options. See Build pipeline source repositories.

Git

XAML PROCESS PARAMETER	TFS 2017 AND NEWER EQUIVALENT	VSTS EQUIVALENT
Clean repository	**Repository** tab, open **Clean** menu, select **true**.	On the **Tasks** tab, in left pane click **Get sources**. Show **Advanced settings**, and then select **Clean**. (We plan to change move this option out of advanced settings.)
Checkout override	You can't specify a commit in the build pipeline, but you can specify one when you manually queue a build.	You can't specify a commit in the build pipeline, but you can specify one when you manually queue a build.

The new build pipeline offers you some new options. See Build pipeline source repositories.

Build

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 Visual Studio Build: How do I build multiple configurations for multiple platforms?
Clean build	Clean
Output location	The Visual Studio Build task builds and outputs files in the same way you do it on your dev machine, in the local workspace. We give you full control of publishing artifacts out of the local workspace on the agent. See Artifacts in Team Build.
Advanced, MSBuild arguments	MSBuild Arguments
Advanced, MSBuild platform	Advanced, MSBuild Architecture
Advanced, Perform code analysis	Use an MSBuild argument such as `/p:RunCodeAnalysis=true`
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 Use a PowerShell script to customize your build process.

> **IMPORTANT**
> In the Visual Studio Build arguments, on the **Visual Studio Version** menu, make sure to select version of Visual Studio that you're using.

The new build pipeline offers you some new options. See Visual Studio Build.

Learn more: Visual Studio Build task (for building solutions), MSBuild task (for building individual projects).

Test

Build process parameters:

1. TF Version Control	
2. Build	
3. Test	
1. Automated tests	**1 set(s) of tests specified.**
1. Test source	- Run tests in test sources matching ***test*.dll;***test*.appx, Targe
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 **Path to publish symbols** argument.

Advanced

Build process parameters:

1. TF Version Control	
2. Build	
3. Test	
4. Publish Symbols	
5. Advanced	
Agent settings	Use agent where Name=* and Tags=[] (MatchExactly)
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 **Options** tab you can specify **Build job timeout in minutes**.

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 Agent capabilities.	A build process asserts demands that are matched with agent capabilities. See Agent capabilities.
Build number format	On the **General** tab, copy your build number format into the **Build number format** field.	On the **General** tab, copy your build number format into the **Build number format** field.
Create work item on failure	On the **Options** tab, select this check box.	On the **Options** tab, enable this option.
Update work items with build number	None	On the **Options** tab you can enable **Automatically link new work in this build**.

The new build pipeline offers you some new options. See:

- Agent capabilities
- Build number format

Retention Policy tab

XAML PROCESS PARAMETER	TFS 2017 AND NEWER, VSTS EQUIVALENT
Retention Policy tab	On the **Retention** tab specify the policies you want to implement.

The new build pipeline offers you some new options. See Build and release retention policies.

Build and release different kinds of apps

In XAML builds you had to create your own custom templates to build different types of apps. In the new build system you can pick from a set of pre-defined templates. The largest and most current set of templates are available on 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 then ever to automatically kick off a deployment after a successful build. Learn more:

- CI/CD Hello world
- Release pipelines
- Triggers

A few examples include:

- Continuous deployment of your app to an Azure web site
- IIS using deployment groups

Other apps and tasks

For more examples of apps you can build and deploy, see Build and deploy your app.

For a complete list of our build, test, and deployment tasks, see Build and release tasks.

Customize your tasks

In XAML builds you created custom XAML tasks. In the new builds, you've got a range of options that begin with easier and lighter-weight approaches.

Get tasks from the Marketplace

Visual Studio Marketplace offers hundreds of extensions that you can install to add tasks that extend your build and deployment capabilities.

Write a script

A major feature of the new build system is its emphasis on using scripts to customize your build 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 pipeline.

TFS 2017 or newer inline PowerShell script

For all these tasks we offer a set of built-in variables, and if necessary, you can define your own variables. See Build variables.

Write a custom task

If necessary, you can write your own custom extensions to custom tasks for your builds and releases.

Reuse patterns

In XAML builds you created custom XAML templates. In the new builds, it's easier to create reusable patterns.

Create a template

If you don't see a template for the kind of app you can start from an empty pipeline and add the tasks you need. After you've got a pattern that you like, you can clone it or save it as a template directly in your web browser. See CI/CD Hello world.

Task groups (TFS 2017 or newer)

In XAML builds, if you change the template, then you also change the behavior of all pipelines based on it. In the new build system, templates don't work this way. Instead, a template behaves as a traditional template. After you create the build pipeline, subsequent changes to the template have no effect on build pipelines.

If you want to create a reusable and automatically updated piece of logic, then create a task group. You can then later modify the task group in one place and cause all the pipelines that use it to automatically be changed.

Q & A

I don't see XAML builds. What do I do?

XAML builds are deprecated. We strongly recommend that you migrate to the new builds as explained above.

If you're not yet ready to migrate, then to enable XAML builds you must connect a XAML build controller to your organization. See Configure and manage your build system.

If you're not yet ready to migrate, then to enable XAML builds:

1. Install TFS 2018.2.
2. Connect your XAML build servers to your TFS instance. See Configure and manage your build system.

How do I add conditional logic to my build process?

Although the new build pipelines are essentially linear, we do give you control of the conditions under which a task runs.

On TFS 2015 and newer: You can select Enabled, Continue on error, or Always run.

On 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

7/17/2018 • 9 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Network Virtualization provides ability to create multiple virtual networks on a shared physical network. Isolated virtual networks can be created using SCVMM Network Virtualization concepts. VMM uses the concept of logical networks and corresponding VM networks to create isolated networks of virtual machines.

- You can create an isolated network of virtual machines that span across different hosts in a host-cluster or a private cloud.
- You can have VMs from different networks residing in the same host machine and still be isolated from each other.
- You can define IP address from the any IP pool of your choice for a VM Network.

See also: Hyper-V Network Virtualization Overview.

To create a virtual network isolated environment:

- Ensure you meet the prerequisite conditions described in this section.
- Set up Network Virtualization using SCVMM. This is a one-time setup task you do not need to repeat.

Follow these steps.

- Decide on the network topology you want to use. You'll specify this when you create the virtual network. The options and steps are described in this section.
- Enable your build-deploy-test scenario as shown in these steps.
- You can perform a range of operations to manage VMs using SCVMM. For examples, see SCVMM deployment.

Prerequisites

- SCVMM Server 2012 R2 or later.
- Window 2012 R2 host machines with Hyper-V set up with at least two physical NICs attached.
- One NIC (perhaps external) with corporate network or Internet access.
- One NIC configured in Trunk Mode with a VLAN ID (such as 991) and routable IP subnets (such as 10.10.30.1/24). You network administrator can configure this.
- All Hyper-V hosts in the host group have the same VLAN ID. This host group will be used for your isolated networks.

Verify the setup is working correctly by following these steps:

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.

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
Reply from 10.10.30.175: bytes=32 time<1ms TTL=128
Reply from 10.10.30.175: bytes=32 time<1ms TTL=128
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.

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 SCVMM deployment.

   > The **SCVMM task** provides a more efficient way capability to perform lab management operations using build and release pipelines. You can manage SCVMM environments, provision isolated virtual networks, and implement build-deploy-test scenarios.

2. In a build or release pipeline, add a new **SCVMM** task.

3. In the **SCVMM** task, select a service connection for the SCVMM server where you want to provision your virtual network and then select **New Virtual machines using Templates/Stored VMs and VHDs** to provision your VMs.

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.

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.

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.

7. Select the **Network Virtualization** option to create the virtualization layer.

8. Based on the topology you would like to create, decide if the network requires an Active Directory VM. For example, to create Topology **2** (AD-backed isolated network), you require an Active directory VM. Select the **Add Active Directory VM** checkbox, enter the AD VM name and the stored VM source. Also enter the static IP address configured in the AD VM source and the DNS suffix.

9. Enter the settings for the **VM Network** and subnet you want to create, and the backing logical network you created in the previous section (Logical Networks). Ensure the VM network name is unique. If possible, append the release name for easier tracking later.

10. In the **Boundary Virtual Machine options** section, set **Create boundary VM for communication with 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.

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.

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 pipeline, 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.

5. Inside the phase, add the tasks you require for deployment and testing.

6. After testing is completed, you can destroy the VMs by using the **Delete VM** task option.

Now you can create release from this release pipeline. Each release will dynamically provision your isolated virtual network and run your deploy and test tasks in the environment. You can find the test results in the release summary. After your tests are completed, you can automatically decommission your environments. You can create as many environments as you need with just a click from the **Build & Release** hub.

Back to list of tasks

See also

- SCVMM deployment
- Hyper-V Network Virtualization Overview

Q&A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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

7/19/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | TFS 2013

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Permissions for build and release functions are primarily set at the object-level for a specific build or release, or for select tasks, at the collection level. For a simplified view of permissions assigned to built-in groups, see Permissions and access.

In addition to permission assignments, you manage security for several resources—such as variable groups, secure files, and deployment groups—by adding users or groups to a role. You grant or restrict permissions by setting the permission state to Allow or Deny, either for a security group or an individual user. For definitions of each build and release permission and role, see Build and release permissions.

Set permissions for build pipelines

1. To set the permissions for all build pipelines, click the Security From the web portal **Build-Release** hub, **Builds** page

To set the permissions for a specific build pipeline, open the context menu for the build and click Security.

2. Choose the group you want to set permissions for, and then change the permission setting to Allow or Deny.

   For example, here we change the permission for Edit build pipeline for the Contributors group to Allow.

3. Save your changes.

Set permissions for release pipelines

1. From the web portal **Build-Release** hub, **Releases** page, open the Security dialog for all release pipelines.

If you want to manage the permissions for a specific release, then open the Security dialog for that release.

2. Choose the group you want to set permissions for, and then change the permission setting to Allow or Deny.

For example, here we deny access to several permissions for the Contributors group.

3. 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 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.

If you want to manage the permissions for a specific variable group, then open the Security dialog for that group.

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.

3. Click **Add**.

Manage task group permissions

Permissions for task groups are subject to a hierarchical model. You use task groups to encapsulate a sequence of tasks already defined in a build or a release pipeline into a single reusable task. You define and manage task groups in the **Task groups** tab of 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.

   If you want to manage the permissions for a specific task group, then open the Security dialog for that group.

2. Add the user or group and then set the permissions you want them to have.

   For example, here we add Raisa and set her permissions to Administer all task groups.

3. Click **Add**.

Set collection-level permissions to administer build resources

1. From the web portal user context, open the admin context by clicking the ⚙ gear Settings icon and choosing **Organization settings** or **Collection settings**.

2. Click **Security**, and then choose the group whose permissions you want to modify.

   Here we choose the Build Administrators group and change the **Use build resources** permission. For a description of each permissions, see Permissions and groups reference, Collection-level permissionss.

3. Save your changes.

Manage permissions for agent queues and service connections

You manage the security for agent pools and service connections by adding users or groups to a role. The method is similar for both agent queues and service connections. You will need to be a member of the Project Administrator group to manage the security for these resources.

> **NOTE**
>
> **Feature availability**: These features are available on VSTS and TFS 2015 and later versions.

For example, here we show how to add a user to the Administrator role for a service connection.

1. From the web portal, click the gear Settings icon to open the project settings admin context.
2. Click **Services**, click the service connection that you want to manage, and then click **Roles**.

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 Settings icon and choose Organization settings or Collection settings to open the collection-level settings admin context.

2. Click **Deployment Pools**, and then open the **Security** dialog for all deployment pools.

   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.

4. Click **Add**.

Related notes

Default build and release permissions

- Default permissions and access

- Permissions and groups reference

Build your GCC C/C++ app

7/19/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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/adventworks/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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.

   - 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 pipeline.

2. In the right panel, select **Empty**, and then click **Apply**. This template allows starting from scratch and adding your own build tasks.

3. For the **Agent queue**:

   - **VSTS:** Select *Hosted Linux* or *Hosted macOS*. 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 pipeline is automatically linked to your repository.

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

VSTS and TFS 2018

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	`make`

TFS 2017.2

1. Search for the **Command Line** task and click **Add** to add it to your build.

2. Click the **Command Line** task and set its field values as follows:

FIELD	VALUE
Display name	`Build C++ application`
Tool	`make`

Finish

1. 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.

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

Save & queue.

3. 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 pipeline 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

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization.

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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.
   - 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 pipeline.

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 pipeline by the template. These are the steps that will automatically run every time you push code changes.

3. For the **Agent queue**:

   - **VSTS:** Select *Hosted Linux*, *Hosted macOS*, or *Hosted VS2017*. This will use a Microsoft-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 pipeline is automatically linked to your repository.

5. Click the **Triggers** tab in the build pipeline. Enable the **Continuous integration** trigger. This will ensure that the build process is automatically triggered every time changes are pushed to your repository.

6. Click **Save & queue** to kick off your first build. On the **Save build pipeline 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.

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 pipeline 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

7/19/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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/adventworks/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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.
   - 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 pipeline.

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 pipeline by the template. These are the tasks that will automatically run every time you push code changes.

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 pipeline is automatically linked to your repository.

5. Click the **Triggers** tab in the build pipeline. 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 pipeline 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.

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 pipeline to meet the needs of your team. To learn more see one of these topics:

- CI builds for Git in VSTS

Build your Universal Windows Platform app

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017.2

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 organization. If you don't have one, you can create one for free. If your team already has one, then make sure you are an administrator of the project you want to use.
- While the simplest way to try this quickstart is to use a VSTS organization, you can also use a TFS server instead of a VSTS organization. Make sure that you have configured a build agent for your 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 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 pipeline that helps your team keep the master branch clean.

1. Create a new build pipeline.
   - 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 pipeline.

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 pipeline by the template. These are the tasks that will automatically run every time you push code changes.

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 pipeline is automatically linked to your repository.

5. Click the **Triggers** tab in the build pipeline. 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 pipeline 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.

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 pipeline to generate production builds.

- Signing UWP package
- Associate package with the store

Troubleshoot Build and Release

7/19/2018 • 10 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

This topic provides general troubleshooting guidance. For specific troubleshooting about .NET Core, see .NET Core troubleshooting

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Run commands locally at the command prompt

It is helpful to narrow whether a build or release failure is the result of a TFS/VSTS product issue (agent or tasks). Build and release failures may also result from external commands.

Check the logs for the exact command-line executed by the failing task. Attempting to run the command locally from the command line may reproduce the issue. It can be helpful to run the command locally from your own machine, and/or log-in to the machine and run the command as the service account.

For example, is the problem happening during the MSBuild part of your build process (for example, are you using either the MSBuild or Visual Studio Build task)? If so, then try running the same MSBuild command on a local machine using the same arguments. If you can reproduce the problem on a local machine, then your next steps are to investigate the MSBuild problem.

Differences between local command prompt and agent

Keep in mind, some differences are in effect when executing a command on a local machine and when a build or release is running on an agent. If the agent is configured to run as a service on Linux, macOS, or Windows, then it is not running within an interactive logged-on session. Without an interactive logged-on session, UI interaction and other limitations exist.

Get logs to diagnose problems

Build and Release logs

Start by looking at the logs in your completed build or release. If they don't provide enough detail, you can make them more verbose:

1. On the **Variables** tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. Queue the build or release.
3. In the explorer tab, view your completed build or release and click the failing task to view its output.
4. If you need a copy of all the logs, click **Download all logs as zip**.

Diagnostic logs

1. On the build summmary page, find the **Queue new build** button, next to it there is a drop down. Click the down arrow and choose **Queue new build with diagnostic logs**.
2. Queue the build.

3. On the build summary page, there will now be a **Diagnostic logs** section. You can download your diagnostic logs per phase. If you would like to download everything you can also choose to **Download all logs as zip**.

> Diagnostic logs are not yet available for releases.

Worker diagnostic logs

You can get the diagnostic log of the completed build that was generated by the worker process on the build agent. Look for the `worker` log file that has the date and time stamp of your completed build. For example, `worker_20160623-192022-utc_6172.log`.

Agent diagnostic logs

Agent diagnostic logs provide a record of how the agent was configured and what happened when it ran. Look for the `agent` log files. For example, `agent_20160624-144630-utc.log`. There are two kinds of agent log files:

- The log file generated when you ran `config.cmd`. This log:
  - Includes this line near the top: `Adding Command: configure`
  - Shows the configuration choices made.
- The log file generated when you ran `run.cmd`. This log:
  - Cannot be opened until the process is terminated.
  - Attempts to connect to your Team Foundation Server or VSTS organization.
  - Shows when each job was run, and how it completed

Both logs show how the agent capabilities were detected and set.

Other logs

Inside the diagnostic logs you will find environment.txt and capabilities.txt.

The environment.txt file has various information about the environment within which your build ran. This includes information like what Tasks are run, whether or not the firewall is enabled, Powershell version info, and some other items. We continually add to this data to make it more useful.

The capabilities file provides a clean way to see all capabilities installed on the build machine that ran your build.

HTTP trace logs

> **Important:** HTTP traces and trace files can contain passwords and other secrets. Do **not** post them on a public sites.

Use built-in HTTP tracing

If your agent is version 2.114.0 or newer, you can trace the HTTP traffic headers and write them into the diagnostic log. Set the `VSTS_AGENT_HTTPTRACE` environment variable before you launch the agent.listener.

```
Windows:
    set VSTS_AGENT_HTTPTRACE=true

macOS/Linux:
    export VSTS_AGENT_HTTPTRACE=true
```

Use full HTTP tracing

Windows

1. Start Fiddler.
2. We recommend you listen only to agent traffic. File > Capture Traffic off (F12)
3. Enable decrypting HTTPS traffic. Tools > Fiddler Options > HTTPS tab. Decrypt HTTPS traffic
4. Let the agent know to use the proxy:

   ```
   set VSTS_HTTP_PROXY=http://127.0.0.1:8888
   ```

5. Run the agent interactively. If you're running as a service, you can set as the environment variable in control panel for the account the service is running as.
6. Restart the agent.

macOS and Linux

Use Charles Proxy (similar to Fiddler on Windows) to capture the HTTP trace of the agent.

1. Start Charles Proxy.
2. Charles: Proxy > Proxy Settings > SSL Tab. Enable. Add URL.
3. Charles: Proxy > Mac OSX Proxy. Recommend disabling to only see agent traffic.

   ```
   export VSTS_HTTP_PROXY=http://127.0.0.1:8888
   ```

4. Run the agent interactively. If it's running as a service, you can set in the .env file. See nix service
5. Restart the agent.

File- and folder-in-use errors

File or folder in use errors are often indicated by error messages such as:

> Access to the path [...] is denied.
> The process cannot access the file [...] because it is being used by another process.
> Access is denied.
> Can't move [...] to [...]

Detect files and folders in use

On Windows, tools like Process Monitor can be to capture a trace of file events under a specific directory. Or, for a snapshot in time, tools like Process Explorer or Handle can be used.

Anti-virus exclusion

Anti-virus software scanning your files can cause file or folder in use errors during a build or release. Adding an anti-virus exclusion for your agent directory and configured "work folder" may help to identify anti-virus software as the interfering process.

MSBuild and /nodeReuse:false

If you invoke MSBuild during your build, make sure to pass the argument `/nodeReuse:false` (short form `/nr:false`). Otherwise MSBuild process(es) will remain running after the build completes. The process(es) remain for some time in anticipation of a potential subsequent build.

This feature of MSBuild can interfere with attempts to delete or move a directory - due to a conflict with the working directory of the MSBuild process(es).

The MSBuild and Visual Studio Build tasks already add `/nr:false` to the arguments passed to MSBuild.

However, if you invoke MSBuild from your own script, then you would need to specify the argument.

MSBuild and /maxcpucount:[n]

By default the build tasks such as MSBuild and Visual Studio Build run MSBuild with the `/m` switch. In some cases this can cause problems such as multiple process file access issues.

Try adding the `/m:1` argument to your build tasks to force MSBuild to run only one process at a time.

File-in-use issues may result when leveraging the concurrent-process feature of MSBuild. Not specifying the argument `/maxcpucount:[n]` (short form `/m:[n]`) instructs MSBuild to use a single process only. If you are using the MSBuild or Visual Studio Build tasks, you may need to specify "/m:1" to override the "/m" argument that is added by default.

Intermittent or inconsistent MSBuild failures

If you are experiencing intermittent or inconsistent MSBuild failures, try instructing MSBuild to use a single-process only. Intermittent or inconsistent errors may indicate that your target configuration is incompatible with the concurrent-process feature of MSBuild. See MSBuild and /maxcpucount:[n]

Process hang

Waiting for Input

A process hang may indicate that a process is waiting for input.

Running the agent from the command line of an interactive logged on session may help to identify whether a process is prompting with a dialog for input.

Running the agent as a service may help to eliminate programs from prompting for input. For example in .Net, programs may rely on the System.Environment.UserInteractive Boolean to determine whether to prompt. When running as a Windows service, the value is false.

Process dump

Analyzing a dump of the process can help to identify what a deadlocked process is waiting on.

WiX project

Building a WiX project when custom MSBuild loggers are enabled, can cause WiX to deadlock waiting on the output stream. Adding the additional MSBuild argument `/p:RunWixToolsOutofProc=true` will workaround the issue.

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 or release to fail with the above error. Apply updates in a controlled fashion to avoid this type of interruption. Before rebooting the agent machine, the agent should first be marked disabled in the pool administration page and let any running build finish.
- Verify the sleep settings are turned off.
- If the agent is running on a virtual machine, avoid any live migration or other VM maintenance operation that may severly 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 or releases not starting

TFS Job Agent not started

This may be characterized by a message in the web console "Waiting for an agent to be requested". Verify the TFSJobAgent (display name: *Visual Studio Team Foundation Background Job Agent*) Windows service is started.

Misconfigured notifcation URL (1.x agent version)

This may be characterized by a message in the web console "Waiting for console output from an agent", and the process eventually times out.

A mismatching notification URL may cause the worker to process to fail to connect to the server. See *Team Foundation Administration Console*, *Application Tier*. The 1.x agent listens to the message queue using the URL that it was configured with. However, when a job message is pulled from the queue, the worker process uses the notification URL to communicate back to the server.

Team Foundation Version Control (TFVC)

Get sources not downloading some files

This may be characterized by a message in the log "All files up to date" from the *tf get* command. Verify the built-in service identity has permission to download the sources. Either the identity *Project Collection Build Service* or *Project Build Service* will need permission to download the sources, depending on the selected authorization scope on General tab of the build pipeline. In the version control web UI, you can browse the project files at any level of the folder hierarchy and check the security settings.

Get sources through Team Foundation Proxy

The easiest way to configure the agent to get sources through a Team Foundation Proxy is set environment variable `TFSPROXY` that point to the TFVC proxy server for the agent's run as user.

```
Windows:
    set TFSPROXY=http://tfvcproxy:8081
    setx TFSPROXY=http://tfvcproxy:8081 // If the agent service is running as NETWORKSERVICE or any
service account you can't easily set user level environment variable
macOS/Linux:
    export TFSPROXY=http://tfvcproxy:8081
```

I need more help. I found a bug. I've got a suggestion. Where do I go?

Get subscription, billing, and technical support

Report any problems on Developer Community.

We welcome your suggestions:

- Propose and vote on ideas on UserVoice.

Default and custom release variables and debugging

7/17/2018 • 11 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

As you compose the tasks for deploying your application into each environment in your DevOps CI/CD processes, 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 view the current values of all variables for a release, and use a default variable to run a release in debug mode.

Custom variables

Custom variables can be defined at various scopes.

- Share values across all of the definitions in a project by using variable groups. Choose a variable group when you need to use the same values across all the definitions, 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 pipeline variables**. Choose a release pipeline variable when you need to use the same value across all the environments and tasks in the release pipeline, and you want to be able to change the value in a single place. You define and manage these variables in the **Variables** tab in a release pipeline.
- Share values across all of the tasks within one specific 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 pipeline.

Using custom variables at project, release pipeline, 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 pipelines. Designate a configuration property to be a secure (secret) variable by selecting the 🔒 (padlock) icon next

to the variable.

> The values of hidden (secret) variables are stored securely on the server and cannot be viewed by users after they are saved. During a deployment, the Release Management service decrypts these values when referenced by the tasks and passes them to the agent over a secure HTTPS channel.

Using custom variables

To use custom variables in your build and release tasks, simply enclose the variable name in parentheses and precede it with a **$** character. For example, if you have a variable named **adminUserName**, you can insert the current value of that variable into a parameter of a task as `$(adminUserName)`.

> **Note:** At present, variables in different groups that are linked to a pipeline in the same scope (such as a release or environment scope) will collide and the result may be unpredictable. Ensure that you use different names for variables across all your variable groups.

You can use custom variables to prompt for values during the execution of a release. For more details, see Approvals.

Define and modify your variables in a script

To define or modify a variable from a script, use the `task.setvariable` logging command. Note that the updated variable value is scoped to the phase being executed, and does not flow across phases or environments.

> **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. Some of the most significant variables are described in the following tables.

System variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.TeamFoundationServerUri	The URL of the Release Management service connection in TFS or VSTS. Use this from your scripts or tasks to call REST APIs on the Release Management service.	https://fabrikam.vsrm.visualstudio.com/	
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or VSTS. 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 project to which this build or release belongs.	Fabrikam	
System.TeamProjectId	The ID of the project to which this build or release belongs.	79f5c12e-3337-4151-be41-a268d2c73344	TFS 2015

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
System.ArtifactsDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.DefaultWorkingDirectory.	C:\agent_work\r1\a	
System.DefaultWorkingDirectory	The directory to which artifacts are downloaded during deployment of a release. The directory is cleared before every deployment if it requires artifacts to be downloaded to the agent. Same as Agent.ReleaseDirectory and System.ArtifactsDirectory.	C:\agent_work\r1\a	
System.WorkFolder	The working directory for this agent, where subfolders are created for every build or release. Same as Agent.RootDirectory and Agent.WorkFolder.	C:\agent_work	
System.Debug	This is the only system variable that can be *set* by the users. Set this to true to run the release in debug mode to assist in fault-finding.	true	

Release variables

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.DefinitionName	The name of the release pipeline to which the current release belongs.	fabrikam-cd	
Release.DefinitionId	The ID of the release pipeline to which the current release belongs.	1	TFS 2015
Release.ReleaseName	The name of the current release.	Release-47	
Release.ReleaseId	The identifier of the current release record.	118	
Release.ReleaseUri	The URI of current release.	vstfs:///ReleaseManagement/Release/118	

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Release.ReleaseDescription	The text description provided at the time of the release.	Critical security patch	
Release.RequestedFor	The display name of identity that triggered the release.	Mateo Escobedo	
Release.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 pipeline.	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

VARIABLE NAME	DESCRIPTION	EXAMPLE	NOT AVAILABLE IN
Agent.Name	The name of the agent as registered with the agent pool. This is likely to be different from the computer name.	fabrikam-agent	
Agent.MachineName	The name of the computer on which the agent is configured.	fabrikam-agent	
Agent.Version	The version of the agent software.	2.109.1	
Agent.JobName	The name of the job that is running, such as Release or Build.	Release	
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.	C:\agent	
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

General artifact variables

For each artifact that is referenced in a release, you can use the following artifact variables. Not all variables are

meaningful for each artifact type. The table below lists the default artifact variables and provides examples of the values that they have depending on the artifact type. If an example is empty, it implies that the variable is not populated for that artifact type.

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/ TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.DefinitionId	The identifier of the build pipeline or repository.	1			fabrikam/asp
Release.Artifacts.{alias}.DefinitionName	The name of the build pipeline or repository.	fabrikam-ci		TFVC: $/fabrikam, Git: fabrikam	fabrikam/asp (master)
Release.Artifacts.{alias}.BuildNumber	The build number or the commit identifier.	20170112.1	20170112.1	TFVC: Changeset 3, Git: 38629c964	38629c964
Release.Artifacts.{alias}.BuildId	The build identifier.	130	130		38629c964d21fe405ef830b7d0220966b82c9e11
Release.Artifacts.{alias}.BuildURI	The URL for the build.	vstfs:///build-release/Build/130			
Release.Artifacts.{alias}.SourceBranch	The path of the branch from which the source was built.	refs/heads/master			
Release.Artifacts.{alias}.SourceBranchName	The name of the branch from which the source was built.	master			
Release.Artifacts.{alias}.SourceVersion	The commit that was built.	bc0044458ba1d9298cdc649cb5dcf013180706f7			
Release.Artifacts.{alias}.Repository.Provider	The type of repository from which the source was built	Git			
Release.Artifacts.{alias}.RequestedForID	The identifier of the account that triggered the build.	2f435d07-769f-4e46 -849d-10d1ab9ba6ab			
Release.Artifacts.{alias}.RequestedFor	The name of the account that requested the build.	Mateo Escobedo			

VARIABLE NAME	DESCRIPTION	TEAM BUILD EXAMPLE	JENKINS/ TEAMCITY EXAMPLE	TFVC/GIT EXAMPLE	GITHUB EXAMPLE
Release.Artifacts.{alias}.Type	The type of artifact source, such as Build.	Build	Jenkins: Jenkins, TeamCity:TeamCity	TFVC: TFVC, Git: Git	GitHub

See also Artifact source alias

Primary artifact variables

You designate one of the artifacts as a primary artifact in a release pipeline. For the designated primary artifact, 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
Build.BuildNumber	Release.Artifacts.{Primary artifact alias}.BuildNumber
Build.BuildId	Release.Artifacts.{Primary artifact alias}.BuildId
Build.BuildURI	Release.Artifacts.{Primary artifact alias}.BuildURI
Build.SourceBranch	Release.Artifacts.{Primary artifact alias}.SourceBranch
Build.SourceBranchName	Release.Artifacts.{Primary artifact alias}.SourceBranchName
Build.SourceVersion	Release.Artifacts.{Primary artifact alias}.SourceVersion
Build.Repository.Provider	Release.Artifacts.{Primary artifact alias}.Repository.Provider
Build.RequestedForID	Release.Artifacts.{Primary artifact alias}.RequestedForID
Build.RequestedFor	Release.Artifacts.{Primary artifact alias}.RequestedFor
Build.Type	Release.Artifacts.{Primary artifact alias}.Type

Using default variables

You can use the default variables in two ways - as parameters to tasks in a release pipeline or in your scripts.

You can directly use a default variable as an input to a task. For example, to pass `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** to a task, you would use `$(Release.Artifacts.ASPNET4.CI.DefinitionName)`.

To use a default variable in your script, you must first replace the `.` in the default variable names with `_`. For example, to print the value of artifact variable `Release.Artifacts.{Artifact alias}.DefinitionName` for the artifact source whose alias is **ASPNET4.CI** in a Powershell script, you would use `$env:RELEASE_ARTIFACTS_ASPNET4_CI_DEFINITIONNAME`.

Note that the original name of the artifact source alias, `ASPNET4.CI`, is replaced by `ASPNET4_CI`.

View the current values of all release variables

In the **Logs** page of a release summary, choose the **Initialize Job** step. The log pane includes a list of all the variables and their values for this release.

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 pipeline.

- 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 pipeline.

> If you get an error related to an Azure RM service connection, see How to: Troubleshoot Azure Resource Manager service connections.

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Troubleshoot Azure Resource Manager service connections

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This topic will help you resolve issues you may encounter when creating a connection to Microsoft Azure using an **Azure Resource Manager** service connection for your DevOps CI/CD processes.

What happens when you create a Resource Manager service connection?

You open the **Add Azure Resource Manager srvice connection** dialog, provide a connection name, and select a subscription from drop-down list of your subscriptions.

When you choose **OK**, the system:

1. Connects to the Azure Active Directory (AAD) tenant for to 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 connection using this application's details

How to troubleshoot errors that may occur

Errors that may occur when the system attempts to create the service connection include:

- Insufficient privileges to complete the operation
- Failed to obtain an access token
- A valid refresh token was not found
- Failed to assign contributor role

Insufficient privileges to complete the operation

This typically occurs when the system attempts to create an application in 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. The account should be an owner, global administrator, or user account administrator.
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. The account should be an owner, global administrator, or user account administrator.
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 connection.

The user is not authorized to add applications in the directory

You must have permission to add integrated applications in the directory. The directory administrator has permission to change this setting, as follows:

1. Choose **Azure Active Directory** in the left navigation bar.
2. Ensure you are editing the appropriate directory corresponding to the user subscription. If not, select **Switch directory** and log in using the appropriate credentials if required.
3. In the **MANAGE** section choose **Users**.
4. Choose **User settings**.
5. In the **App registrations** section, change **Users can register applications** to **Yes**.

Failed to obtain an access token or A valid refresh token was not found

These errors typically occur when your session has expired.

To resolve these issues:

- Sign out of VSTS or TFS.
- Open an InPrivate or incognito browser window and navigate to https://visualstudio.microsoft.com/team-services/.
- If you are prompted to sign out, do so.
- Sign in using the appropriate credentials.
- Choose the organization you want to use from the list.

- Select the project you want to add the service connection to.
- Create the service connection you need by opening the **Settings** page, selecting the **Services** tab, choosing **New service connection**, and selecting **Azure Resource Manager**.

Failed to assign Contributor role

This error typically occurs when you do not have **Write** permission for the selected Azure subscription when the system attempts to assign the **Contributor** role.

To resolve this issue, ask the subscription administrator to configure your identity in an **Admin Access** role.

Help and support

- See our troubleshooting page.
- Report any problems on Developer Community, make suggestions on UserVoice, get advice on Stack Overflow, and get support via our Support page.

Build and release tasks

7/9/2018 • 10 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015 | Previous versions (XAML builds)

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Build

TASK	VERSIONS
Android Build - deprecated. Use Gradle	VSTS, TFS 2015 RTM and newer
Android Signing. Sign and align Android APK files	VSTS, TFS 2015 RTM and newer
Ant. Build with Apache Ant	VSTS, TFS 2015 RTM and newer
CMake. Build with the CMake cross-platform build system	VSTS, TFS 2015 RTM and newer
Docker. Build, tag, push, or run Docker images, or run a Docker command.	VSTS
Docker Compose. Build, push or run multi-container Docker applications.	VSTS
Go. Get, build, test a go application, or run a custom go command.	VSTS
Gradle. Build using a Gradle wrapper script	VSTS, TFS 2015 RTM and newer
Grunt. The JavaScript Task Runner	VSTS, TFS 2015.3 and newer
Gulp. Node.js streaming task based build system	VSTS, TFS 2015 RTM and newer
Index Sources & Publish Symbols. Index your source code and publish symbols to a file share	VSTS, TFS 2015 RTM and newer

TASK	VERSIONS
Jenkins Queue Job. Queue a job on a Jenkins server	VSTS, TFS 2017 RTM and newer
Maven. Build with Apache Maven	VSTS, TFS 2015 RTM and newer
MSBuild. Build with MSBuild	VSTS, TFS 2015 RTM and newer
.NET Core CLI. Build, test, package, or publish a dotnet application, or run a custom dotnet command.	VSTS, TFS 2018
Publish Build Artifacts. Publish Build artifacts to the server or a file share	TFS 2015 RTM. Deprecated on VSTS and newer versions of TFS.
SonarQube - Begin Analysis. Fetch the Quality Profile from SonarQube to configure the analysis	VSTS, TFS 2015.3 and newer
SonarQube - End Analysis. Finish the analysis and upload the results to SonarQube	VSTS, TFS 2015.3 and newer
Visual Studio Build. 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. 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

TASK	VERSIONS
Azure Network Load Balancer .Connect/Disconnect an Azure virtual machine's network interface to a Load Balancer's backend address pool	VSTS
Bash . Run a Bash script on macOS, Linux, or Windows	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
cURL Upload Files . Use cURL to upload files with supported protocols. (FTP, FTPS, SFTP, HTTP, and more)	VSTS, TFS 2015 RTM and newer
Decrypt File . A thin utility task for file decryption using OpenSSL.	VSTS
Delay . Pause execution of the process for a fixed delay time.	VSTS
Delete Files . Delete files or folders.	VSTS, TFS 2015.3 and newer
Download Build Artifacts . Download build artifacts.	VSTS
Download Package . Download a package from a Package Management feed in VSTS or TFS. Requires the Package Management extension.	VSTS
Download Secure File . Download a secure file to a temporary location on the build or release agent.	VSTS
Extract Files . Extract files from archives (.zip, .jar, .war, .ear, .tar, .7z., and others) to a target folder.	VSTS, TFS 2017 and newer

TASK	VERSIONS
FTP Upload. Upload files to a remote machine using the File Transfer Protocol (FTP), or securely with FTPS.	VSTS, TFS 2017 and newer
Install Apple Certificate. Install an Apple certificate required to build on a macOS agent.	VSTS, TFS 2018
Install Apple Provisioning Profile. Install an Apple provisioning profile required to build on a macOS agent.	VSTS, TFS 2018
Install SSH Key. Install an SSH key prior to a build or release	VSTS
Invoke HTTP REST API. Invoke an HTTP API and parse the response.	VSTS
Jenkins Download Artifacts. Download artifacts produced by a Jenkins job	VSTS
Manual Intervention. Pause an active deployment within an environment, typically to perform some manual steps or actions, and then continue the automated deployment tasks.	VSTS
PowerShell. Run a PowerShell script	VSTS, TFS 2015 RTM and newer
Publish Build Artifacts. Publish Build artifacts to the server or a file share	VSTS, TFS 2015.3 and newer
Publish To Azure Service Bus. Send a message to an Azure Service Bus using a service connection and without using an agent.	VSTS
Python Script. Run a Python script.	VSTS
Query Azure Monitor Alerts. Observe the configured Azure monitor rules for active alerts.	VSTS
Query Work Items. Ensure the number of matching items returned by a work item query in within the configured thresholds.	VSTS
Service Fabric PowerShell. Runs any PowerShell command or script in a PowerShell session that has a Service Fabric cluster connection initialized.	VSTS

TASK	VERSIONS
Shell Script. Run a shell script using bash	VSTS, TFS 2015 RTM and newer
Update Service Fabric App Versions. Automatically updates the versions of a packaged Service Fabric application	VSTS, TFS 2017 and newer

Test

TASK	VERSIONS
App Center Test. Test mobile app packages with Visual Studio App Center	VSTS, TFS 2015.3 and newer
Cloud-based Apache JMeter Load Test. Runs the Apache JMeter load test in cloud	VSTS, TFS 2015 RTM and newer
Cloud-based Load Test. Runs the load test in cloud, with VSTS	VSTS, TFS 2015 RTM and newer
Cloud-based Web Performance Test. Runs the quick web performance test in cloud, with VSTS	VSTS, TFS 2015 RTM and newer
Publish Code Coverage Results. Publish code coverage results to VSTS/TFS	VSTS, TFS 2015.3 and newer
Publish Test Results. Publish Test Results to VSTS/TFS	VSTS, TFS 2015 RTM and newer
Run Functional Tests. Run Coded UI/Selenium/Functional tests on a set of machines (using Test Agent)	VSTS, TFS 2015.3 and newer
Visual Studio Test version 2 Visual Studio Test version 1. Run unit and functional tests (Selenium, Appium, Coded UI test, etc.) using the Visual Studio Test runner. Test frameworks that have a Visual Studio test adapter such as xUnit, NUnit, Chutzpah, etc. can also be run.	VSTS, TFS 2015 RTM and newer
Visual Studio Test Agent Deployment. Deploy and configure Test Agent to run tests on a lab machine group	VSTS, TFS 2015 RTM and newer

Package

TASK	VERSIONS
Conda Environment. Create and activate a Conda environment	VSTS
CocoaPods. CocoaPods is the dependency manager for Swift and Objective-C Cocoa projects. Runs pod install	VSTS, TFS 2015 RTM and newer
npm. Install npm packages	VSTS, TFS 2015 RTM and newer
npm Authenticate. Don't use this task if you're also using the npm task. Provides npm credentials to an .npmrc file in your repository for the scope of the build.	VSTS, TFS 2015 RTM and newer
NuGet Installer. Installs and updates missing NuGet packages	VSTS, TFS 2015 RTM and newer
NuGet Packager. Creates nupkg outputs from csproj or nuspec files	VSTS, TFS 2015.3 and newer
NuGet Publisher. Uploads nupkg files to a nuget server	VSTS, TFS 2015.3 and newer
PyPI Publisher. Publish a Python package to PyPI	VSTS
Xamarin Component Restore. - deprecated. See Updating component references to NuGet	VSTS, TFS 2015 RTM and newer

Deploy

TASK	VERSIONS
App Center Distribute. Distribute app builds to testers and users via App Center	VSTS, TFS 2017 and newer
Azure App Service Deploy. Update Azure App Service using Web Deploy / Kudu REST APIs	VSTS, TFS 2017 and newer
Azure App Service Manage. Start, Stop, Restart or Slot swap for an Azure App Service	VSTS, TFS 2017 and newer
Azure CLI. Run a shell or batch script containing Azure CLI commands against an Azure subscription	VSTS, TFS 2017 and newer

TASK	VERSIONS
Azure File Copy. Copy files to Azure blob or VM(s)	VSTS, TFS 2015.3 and newer
Azure Key Vault. Incorporate secrets from an Azure Key Vault into a release pipeline	VSTS
Azure Monitor Alerts. Configure alerts on available metrics for an Azure resource	VSTS
Azure MySQL Deployment. Run your scripts and make changes to your Azure DB for MySQL.	VSTS
Azure PowerShell. Run a PowerShell script within an Azure environment	VSTS, TFS 2015 RTM and newer
Azure Resource Group Deployment. Deploy, start, stop, delete Azure Resource Groups	VSTS, TFS 2015.3 and newer
Azure SQL Database Deployment. Deploy an Azure SQL database using DACPAC or run scripts using SQLCMD	VSTS, TFS 2015.3 and newer
Azure VMSS Deployment. Deploy a virtual machine scale set image.	VSTS
Chef. Deploy to Chef environments by editing environment attributes	VSTS
Chef Knife. Run Scripts with knife commands on your chef workstation	VSTS
Copy Files Over SSH. Copy files from source folder to target folder on a remote machine over SSH	VSTS, TFS 2017 and newer
Docker. Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry	VSTS, TFS 2017 and newer
Docker Compose. Build, push or run multi-container Docker applications.	VSTS
Helm Deploy. Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.	VSTS

TASK	VERSIONS
IIS Web App Deploy. Deploy a Website or Web Application using WebDeploy	VSTS
IIS Web App Manage. Create or update a Website, Web App, Virtual Directories, and Application Pool	VSTS
Kubernetes. Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.	VSTS
Packer Build. Build a machine image using Packer.	VSTS
PowerShell on Target Machines. Execute PowerShell scripts on remote machine(s)	VSTS, TFS 2015 RTM and newer
Service Fabric Application Deployment. Deploy a Service Fabric application to a cluster	VSTS, TFS 2017 and newer
Service Fabric Compose Deploy. Deploy a Service Fabric application to a cluster using a compose file	VSTS
SSH. Run shell commands or a script on a remote machine using SSH	VSTS, TFS 2017 and newer
SQL Server Database Deploy. Deploy a SQL Server database using DACPAC or SQL scripts	VSTS
Windows Machine File Copy. Copy files to remote machine(s)	VSTS, TFS 2015 RTM and newer

Tool

TASK	VERSIONS
Go Tool Installer. Finds or downloads a specific version of the Go tool into the tools cache and adds it to the PATH	VSTS
Helm Tool Installer. Install Helm and Kubernetes on agent machine.	VSTS
Java Tool Installer. Acquires a specific version of Java from a user supplied Azure blob, a location in the souce 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

TASK	VERSIONS
.NET Core Tool Installer. Acquires a specific version of .NET Core and adds it to the PATH. Use the task to change the Core version for subsequent tasks.	VSTS, TFS 2018
Node Tool Installer. Finds or downloads and caches the specified version of Node.js and adds it to the PATH	VSTS
Use Python Version. Selects a version of Python to run on an agent. Optionally adds it to PATH.	VSTS
Use Ruby Version. Selects a version of Ruby to run on an agent. Optionally adds it to PATH.	VSTS
Visual Studio Test Platform Installer. Acquires the Visual Studio Test Platform from nuget.org or the tools cache.	VSTS

To learn more about tool installer tasks, see Tool installers.

Open source

These tasks are open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn step-by-step how to build my app?

Build your app

Can I add my own build tasks?

Yes: Add a build task

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/19/2018 • 6 minutes 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.

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Restore NuGet packages

Demands

None

YAML snippet

```
# .NET Core
# Restore NuGet packages.
- task: DotNetCoreCLI@2
  inputs:
    command: 'restore'
    projects: '**/*.csproj'
    #verbosityRestore: 'detailed' # Options: quiet, minimal, normal, detailed, diagnostic
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Path to project(s)	(Optional) The path to the csproj file(s) to use. You can use wildcards (e.g. **/.csproj for all .csproj files in all subfolders).
Verbosity	(Required) Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Pack NuGet packages

Demands

None

YAML snippet

```
# .NET Core
# Pack NuGet packages.
- task: DotNetCoreCLI@2
  inputs:
    command: 'pack'
    configuration: 'release'
    #packagesToPack: '**/*.csproj' # Required when command == pack
    #packDirectory: '$(build.artifactStagingDirectory)' # Optional
    #nobuild: false # Optional
    #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
    #versionEnvVar: # Required when versioningScheme == byEnvVar
    #majorVersion: '1' # Required when versioningScheme == byPrereleaseNumber
    #minorVersion: '0' # Required when versioningScheme == byPrereleaseNumber
    #patchVersion: '0' # Required when versioningScheme == byPrereleaseNumber
    #buildProperties: # Optional
    #verbosityPack: 'detailed' # Options: quiet, minimal, normal, detailed, diagnostic
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Configuration to Package	(Optional) When using a csproj file this specifies the configuration to package
Package Folder	(Optional) Folder where packages will be created. If empty, packages will be created alongside the csproj file.
Do not build	(Optional) Don't build the project before packing. Corresponds to the `--no-build` command line parameter.
Automatic package versioning	(Required) Cannot be used with include referenced projects. If you choose 'Use the date and time', this will generate a SemVer-compliant version formatted as `X.Y.Z-ci-datetime` where you choose X, Y, and Z. If you choose 'Use an environment variable', you must select an environment variable and ensure it contains the version number you want to use. If you choose 'Use the build number', this will use the build number to version your package. **Note:** Under Options set the build number format to be '$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.r)'.
Environment variable	(Required) Enter the variable name without $, $env, or %.
Major	(Required) The 'X' in version X.Y.Z
Minor	(Required) The 'Y' in version X.Y.Z
Patch	(Required) The 'Z' in version X.Y.Z
Additional build properties	(Optional) Specifies a list of token = value pairs, separated by semicolons, where each occurrence of $token$ in the .nuspec file will be replaced with the given value. Values can be strings in quotation marks.

ARGUMENT	DESCRIPTION
Verbosity	(Required) Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demands

None

YAML snippet

```
# .NET Core
# Push NuGet packages.
- task: DotNetCoreCLI@2
  inputs:
    command: 'push'
    #nuGetFeedType: 'internal' # Required when command == push. Options: internal, external
    #packagesToPush: '$(build.artifactStagingDirectory)/*.nupkg' # Required when command == push
    #publishVstsFeed: # Required when command == push && NuGetFeedType == internal
    #publishFeedCredentials: # Required when command == push && NuGetFeedType == external
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Target feed location	(Required) Use 'internal' for this organization/collection. Use 'external' for an external NuGet server (including other accounts/collections).
Path to NuGet package(s) to publish	(Required) The pattern to match or path to nupkg files to be uploaded. Multiple patterns can be separated by a semicolon, and you can make a pattern negative by prefixing it with '-:'. Example: `**\*.nupkg;-:**\*.Tests.nupkg`
Target feed	(Required) Select a feed hosted in this organization. You must have Package Management installed and licensed to select a feed here.
NuGet server	(Required) The NuGet service connection that contains the external NuGet server's credentials.
CONTROL OPTIONS	

Custom NuGet command

YAML snippet

```
# .NET Core
# Custom NuGet command.
- task: DotNetCoreCLI@2
  inputs:
    command: custom
    projects: '**/.csproj'
    custom: 'Enter your custom NuGet command here'
    arguments: '--configuration release --output $(build.artifactStagingDirectory)'
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.
Path to project(s)	(Optional) The path to the csproj file(s) to use. You can use wildcards (e.g. **/.csproj for all .csproj files in all subfolders).
Custom command	(Required) The command to pass to dotnet.exe for execution.
Arguments	(Optional) Arguments to the selected command. For example, build configuration, output folder, runtime. The arguments depend on the command selected.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Why is my build or publish step failing to restore packages?

Most `dotnet` commands, including `build` and `publish`, include an implicit `restore` step. This will fail against authenticated feeds, even if you ran a successful `dotnet restore` in an earlier step, because the earlier step will have cleaned up the credentials it used.

To fix this issue, add the `--no-restore` flag to the Arguments textbox.

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project-the location of its packages-is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add 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 Seting up a solution to get dependencies

What other kinds of apps can I build?

Build your app

What other kinds of build tasks are available?

Specify your build tasks

How do we protect our codebase from build breaks?

- Git: Improve code quality with branch policies with an option to require that code builds before it can be merged to a branch. This option is not available for GitHub repos.
- TFVC: Use gated check-in.

How do I modify other parts of my build pipeline?

- Specify your build tasks to run tests, scripts, and a wide range of other processes.
- Specify build options such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- Specify the repository to pick the source of the build and modify options such as how the agent workspace is cleaned.
- Set build triggers to modify how your CI builds run and to specify scheduled builds.
- Specify build retention policies to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent jobs. See Concurrent build and release jobs in VSTS.

How do I see what has changed in my build pipeline?

View the change history of your build pipeline

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 CLI

7/19/2018 • 5 minutes to read • Edit Online

Build, test, package, or publish a dotnet application, or run a custom dotnet command. For package commands, supports NuGet.org and authenticated feeds like Package Management and MyGet.

YAML snippet

```
# .NET Core
# Build, test, package, or publish a dotnet application, or run a custom dotnet command. For package commands,
supports NuGet.org and authenticated feeds like Package Management and MyGet.
- task: DotNetCoreCLI@2
  inputs:
    #command: 'build' # Options: build, push, pack, publish, restore, run, test, custom
    #publishWebProjects: true # Required when command == Publish
    #projects: # Optional
    #custom: # Required when command == Custom
    #arguments: # Optional
    #publishTestResults: true # Optional
    #zipAfterPublish: true # Optional
    #modifyOutputPath: true # Optional
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    restoreDirectory:
    #verbosityRestore: 'Detailed' # Options: -, quiet, minimal, normal, detailed, diagnostic
    #packagesToPush: '$(Build.ArtifactStagingDirectory)/*.nupkg' # Required when command == Push
    #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
    #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
    #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
    #packagesToPack: '**/*.csproj' # Required when command == Pack
    #configuration: '$(BuildConfiguration)' # Optional
    #packDirectory: '$(Build.ArtifactStagingDirectory)' # Optional
    #nobuild: false # Optional
    #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
    #versionEnvVar: # Required when versioningScheme == ByEnvVar
    #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
    #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #buildProperties: # Optional
    #verbosityPack: 'Detailed' # Options: -, quiet, minimal, normal, detailed, diagnostic
    workingDirectory:
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) The dotnet command to run. Select 'Custom' to add arguments or use a command not listed here.

ARGUMENT	DESCRIPTION
Publish Web Projects	(Required) If true, the task will try to find the web projects in the repository and run the publish command on them. Web projects are identified by presence of either a web.config file or wwwroot folder in the directory.
Path to project(s)	(Optional) The path to the csproj file(s) to use. You can use wildcards (e.g. **/.csproj for all .csproj files in all subfolders).
Custom command	(Required) The command to pass to dotnet.exe for execution.
Arguments	(Optional) Arguments to the selected command. For example, build configuration, output folder, runtime. The arguments depend on the command selected.
Publish test results	(Optional) Enabling this option will generate a test results TRX file in `$(Agent.TempDirectory)` and results will be published to the server. This option appends `--logger trx --results-directory $(Agent.TempDirectory)` to the command line arguments.
Zip Published Projects	(Optional) If true, folder created by the publish command will be zipped.
Add project name to publish path	(Optional) If true, folders created by the publish command will have project file name prefixed to their folder names when output path is specified explicitly in arguments. This is useful if you want to publish multiple projects to the same folder.
Feeds to use	(Required) You can either select a feed from VSTS and/or NuGet.org here, or commit a nuget.config file to your source code repository and set its path here.
Use packages from this VSTS/TFS feed	(Required) Include the selected feed in the generated NuGet.config. You must have Package Management installed and licensed to select a feed here.
Use packages from NuGet.org	(Required) Include NuGet.org in the generated NuGet.config.
Path to NuGet.config	(Required) The NuGet.config in your repository that specifies the feeds from which to restore packages.
Credentials for feeds outside this organization/collection	(Optional) Credentials to use for external registries located in the selected NuGet.config. For feeds in this organization/collection, leave this blank; the build's credentials are used automatically.
Disable local cache	(Required) Prevents NuGet from using packages from local machine caches.
Destination directory	(Required) Specifies the folder in which packages are installed. If no folder is specified, packages are restored into the default NuGet package cache.

ARGUMENT	DESCRIPTION
Verbosity	(Required) Specifies the amount of detail displayed in the output.
Path to NuGet package(s) to publish	(Required) The pattern to match or path to nupkg files to be uploaded. Multiple patterns can be separated by a semicolon, and you can make a pattern negative by prefixing it with '-:'. Example: **/*.nupkg;-:**/*.Tests.nupkg
Target feed location	(Required) undefined
Target feed	(Required) Select a feed hosted in this organization. You must have Package Management installed and licensed to select a feed here.
NuGet server	(Required) The NuGet service connection that contains the external NuGet server's credentials.
Path to csproj or nuspec file(s) to pack	(Required) Pattern to search for csproj or nuspec files to pack. You can separate multiple patterns with a semicolon, and you can make a pattern negative by prefixing it with '-:'. Example: **/*.csproj;-:**/*.Tests.csproj
Configuration to Package	(Optional) When using a csproj file this specifies the configuration to package
Package Folder	(Optional) Folder where packages will be created. If empty, packages will be created alongside the csproj file.
Do not build	(Optional) Don't build the project before packing. Corresponds to the `--no-build` command line parameter.
Automatic package versioning	(Required) Cannot be used with include referenced projects. If you choose 'Use the date and time', this will generate a SemVer-compliant version formatted as `X.Y.Z-ci-datetime` where you choose X, Y, and Z. If you choose 'Use an environment variable', you must select an environment variable and ensure it contains the version number you want to use. If you choose 'Use the build number', this will use the build number to version your package. **Note:** Under Options set the build number format to be '$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.r)'.
Environment variable	(Required) Enter the variable name without $, $env, or %.
Major	(Required) The 'X' in version X.Y.Z
Minor	(Required) The 'Y' in version X.Y.Z
Patch	(Required) The 'Z' in version X.Y.Z
Additional build properties	(Optional) Specifies a list of token = value pairs, separated by semicolons, where each occurrence of $token$ in the .nuspec file will be replaced with the given value. Values can be strings in quotation marks.

ARGUMENT	DESCRIPTION
Verbosity	(Required) Specifies the amount of detail displayed in the output.
Working Directory	(Required) Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is $(System.DefaultWorkingDirectory)
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Build: Android build (deprecated; use Gradle)

7/9/2018 • 2 minutes 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 Microsoft-hosted agents), you must use the `gradlew.bat` wrapper. Agents on Linux or macOS can use the `gradlew` shell script. See The Gradle Wrapper.
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 `build` See Gradle command line.
ANDROID VIRTUAL DEVICE (AVD) OPTIONS	
Name	Name of the AVD to be started or created. **Note:** You must deploy your own agent 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 `android-19`
AVD Device	(Optional) Device pipeline to use. Can be a device index or id. The default value is `Nexus 5`
AVD ABI	The Application Binary Interface to use for the AVD. The default value is `default/armeabi-v7a` See ABI Management.
Overwrite Existing AVD	Select this check box if an existing AVD with the same name should be overwritten.
Create AVD Optional Arguments	Provide any options to pass to the `android create avd` command. See Android Command Line.
EMULATOR OPTIONS	
Start and Stop Android Emulator	Check if you want the emulator to be started and stopped when Android Build task finishes. > **Note:** You must deploy your own agent to use this option. You cannot use a 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 `300` 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 `emulator` command. The default value is `-no-snapshot-load -no-snapshot-save`
Delete AVD	Check if you want the AVD to be deleted upon completion.
CONTROL OPTIONS	

Related tasks

Android Signing

Build: Android signing

7/9/2018 • 2 minutes 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

YAML snippet

```
# Android Signing
# Sign and align Android APK files
- task: AndroidSigning@2
  inputs:
    #apkFiles: '**/*.apk'
    #jarsign: true # Optional
    #jarsignerKeystoreFile: # Required when jarsign == True
    #jarsignerKeystorePassword: # Optional
    #jarsignerKeystoreAlias: # Optional
    #jarsignerKeyPassword: # Optional
    #jarsignerArguments: '-verbose -sigalg MD5withRSA -digestalg SHA1' # Optional
    #zipalign: true # Optional
    #zipalignFile: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
APK Files	Relative path from the repo root to the APK(s) you want to sign. You can use wildcards to specify multiple files. For example: • `outputs\apk\*.apk` to sign all .APK files in the outputs\apk\ subfolder • `**\\bin\\*.apk` to sign all .APK files in all bin subfolders
SIGNING OPTIONS	
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.

ARGUMENT	DESCRIPTION
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 **Decrypt File** task to decrypt the file during the build.
Keystore Password	Enter the password for the provided keystore file. **Important:** We recommend that you put this value in a secret variable.
Alias	Enter the alias that identifies the public/private key pair to be used in the keystore file.
Key Password	Enter the key password for the alias and keystore file. **Important:** We recommend that you put this value in a secret variable.
Jarsigner Arguments	Provide any options to pass to the jarsigner command line. Default is `-verbose -sigalg MD5withRSA -digestalg SHA1` See jarsigner documentation.
ZIPALIGN OPTIONS	
Zipalign	Select if you want to zipalign your package. This reduces the amount of RAM consumed by an app.
Zipalign Location	(Optional) The location of the zipalign executable used during signing. Defaults to the zipalign found in the Android SDK version folder your application builds against.
CONTROL OPTIONS	

Related tasks

Android Build

Build: Ant

7/9/2018 • 3 minutes 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

YAML snippet

```
# Ant
# Build with Apache Ant
- task: Ant@1
  inputs:
    #buildFile: 'build.xml'
    #options: # Optional
    #targets: # Optional
    #publishJUnitResults: true
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #codeCoverageToolOptions: 'None' # Optional. Options: none, cobertura, jaCoCo
    #codeCoverageClassFilesDirectories: '.' # Required when codeCoverageToolOptions != None
    #codeCoverageClassFilter: # Optional
    #codeCoverageSourceDirectories: # Optional
    #codeCoverageFailIfEmpty: false # Optional
    #antHomeDirectory: # Optional
    #javaHomeOption: 'JDKVersion' # Options: jDKVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkUserInputDirectory: # Required when javaHomeOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
```

Arguments

ARGUMENT	DESCRIPTION
Ant Build File	Relative path from the repository root to the Ant build file. For more information about build files, see Using Apache Ant.

ARGUMENT	DESCRIPTION
Options	Options that you want to pass to the Ant command line. You can provide your own properties (for example, `-DmyProperty=myPropertyValue`) and also use built-in variables (for example, `-DcollectionId=$(system.collectionId)`). Alternatively, the built-in variables are already set as environment variables during the build and can be passed directly (for example, `-DcollectionIdAsEnvVar=%SYSTEM_COLLECTIONID%`). See Running Apache Ant.
Target(s)	Target(s) for Ant to execute for this build. See Using Apache Ant Targets.
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, `**/TEST-*.xml` 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	Select the code coverage tool you want to use. If you are using the Microsoft-hosted agents, then the tools are set up for you. If you are using on-premises Windows agent, then if you select: • JaCoCo, make sure jacocoant.jar is available in lib folder of Ant installation. See JaCoCo. • Cobertura, set up an environment variable COBERTURA_HOME pointing to the Cobertura .jar files location. See Cobertura. After you select one of these tools, the following arguments appear.
Class Files Directories	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.

ARGUMENT	DESCRIPTION
Class Inclusion/Exclusion Filters	Specify a comma-separated list of filters to include or exclude classes from collecting code coverage. For example: `+:com.,+:org.,-:my.app.`
Source Files Directories	Specify a comma-separated list of relative paths from the Ant build file to your source directories. Code coverage reports will use these paths to highlight source code. For example: src/java,src/Test.
ADVANCED	
Set ANT_HOME Path	If set, overrides any existing ANT_HOME environment variable with the given path.
Set JAVA_HOME by JDK Version	Choose which JDK level will be used to run Ant. Will attempt to find JDK version and assign JAVA_HOME before running Ant.
Set JAVA_HOME by Path	Directory on build agent where the JDK is located.
JDK Architecture	Optionally supply the architecture (x86, x64) of the JDK.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Build with the CMake cross-platform build system

Demands

cmake

YAML snippet

```
# CMake
# Build with the CMake cross-platform build system
- task: CMake@1
  inputs:
    #workingDirectory: 'build' # Optional
    #cmakeArgs: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Working Directory	Working directory when CMake is run. The default value is `build`. If you specify a relative path, then it is relative to your repo. For example, if you specify `build`, the result is the same as if you specified `$(Build.SourcesDirectory)\build`. You can also specify a full path outside the repo, and you can use variables. For example: `$(Build.ArtifactStagingDirectory)\build` If the path you specify does not exist, CMAke creates it.
Arguments	Arguments that you want to pass to CMake.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

How do I enable CMake for Microsoft-hosted agents?

The Microsoft-hosted agents have CMake installed, but you must manually add the capability to use the CMake build task.

1. Open the Agent Pools control panel tab:
   - VSTS: `https://{your_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?
2. 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_organization}.visualstudio.com/_admin/_AgentPool`
   - TFS 2018: `https://{your_server}/DefaultCollection/_admin/_AgentPool`
   - TFS 2017: `https://{your_server}/tfs/DefaultCollection/_admin/_AgentPool`
   - TFS 2015: `http://{your_server}:8080/tfs/_admin/_AgentPool`

   The TFS URL doesn't work for me. How can I get the correct URL?
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: Docker

7/9/2018 • 3 minutes to read • Edit Online

Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry.

YAML snippet

```
# Docker
# Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure
Container registry.
- task: Docker@1
  inputs:
    #containerregistrytype: 'Azure Container Registry' # Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscriptionEndpoint: # Optional
    #azureContainerRegistry: # Optional
    #command: 'Build an image' # Options: build An Image, tag Image, push An Image, run An Image
    #dockerFile: '**/Dockerfile' # Required when command == Build An Image
    #arguments: # Optional
    #useDefaultContext: true # Optional
    #buildContext: # Optional
    #pushMultipleImages: false # Optional
    #tagMultipleImages: false # Optional
    #imageName: '$(Build.Repository.Name):$(Build.BuildId)' # Required when command == Build An Image ||
Command == Run An Image || PushMultipleImages == False || TagMultipleImages == False
    #imageNamesPath: # Required when tagMultipleImages == True || PushMultipleImages == True
    #qualifyImageName: true # Optional
    #includeSourceTags: false # Optional
    #includeLatestTag: false # Optional
    #imageDigestFile: # Optional
    #containerName: # Optional
    #ports: # Optional
    #volumes: # Optional
    #envVars: # Optional
    #workingDirectory: # Optional
    #entrypointOverride: # Optional
    #containerCommand: # Optional
    #runInBackground: true # Optional
    #restartPolicy: 'no' # Required when command == Run An Image && RunInBackground == True# Options: no,
onFailure, always, unlessStopped
    #maxRestartRetries: # Optional
    #dockerHostEndpoint: # Optional
    #enforceDockerNamingConvention: true # Optional
    #memoryLimit: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.

ARGUMENT	DESCRIPTION
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Action	(Required) Select a Docker action.
Docker File	(Required) Path to the Docker file to use. Must be within the Docker build context.
Build Arguments	(Optional) Build-time variables for the Docker file. Specify each name=value pair on a new line.
Use Default Build Context	(Optional) Set the build context to the directory that contains the Docker file.
Build Context	(Optional) Path to the build context.
Image Name	(Required) Name of the Docker image to build, push, or run.
Image Names Path	(Required) Path to a text file that contains the names of the Docker images to tag or push. Each image name is contained on its own line.
Qualify Image Name	(Optional) Qualify the image name with the Docker registry connection's hostname if not otherwise specified.
Additional Image Tags	(Optional) Additional tags for the Docker image being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing the Docker image.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing the Docker image.
Image Digest File	(Optional) Path to a file that is created and populated with the full image repository digest of the Docker image that was pushed.
Container Name	(Optional) Name of the Docker container to run.
Ports	(Optional) Ports in the Docker container to publish to the host. Specify each host-port:container-port binding on a new line.
Volumes	(Optional) Volumes to mount from the host. Specify each host-dir:container-dir on a new line.
Environment Variables	(Optional) Environment variables for the Docker container. Specify each name=value pair on a new line.
Working Directory	(Optional) The working directory for the Docker container.

ARGUMENT	DESCRIPTION
Entrypoint Override	(Optional) Override the default entrypoint for the Docker container.
Command	(Optional) Command to run in the Docker container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the Docker container in the background.
Restart Policy	(Required) Select a restart policy.
Maximum Restart Retries	(Optional) The maximum number of restart retries the Docker daemon attempts.
Command	(Required) Docker command to execute, with arguments. For example, 'rmi -f image-name' to force remove an image.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
Force image name to follow Docker naming convention	(Optional) If enabled docker image name will be modified to follow Docker naming convention. Converts upper case character to lower case and removes spaces in image name.
Working Directory	(Optional) Working directory for the Docker command.
Memory limit	(Optional) The maximum amount of memory available to the container as a integer with optional suffixes like '2GB'.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Build: Docker Compose

7/9/2018 • 3 minutes to read • Edit Online

Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container registry.

YAML snippet

```
# Docker Compose
# Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container
registry.
- task: DockerCompose@0
  inputs:
    #containerregistrytype: 'Azure Container Registry' # Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscription: # Optional
    #azureContainerRegistry: # Optional
    #dockerComposeFile: '**/docker-compose.yml'
    #additionalDockerComposeFiles: # Optional
    #dockerComposeFileArgs: # Optional
    #projectName: '$(Build.Repository.Name)' # Optional
    #qualifyImageNames: true # Optional
    #action: 'Run a Docker Compose command' # Options: build Services, push Services, run Services, run A
Specific Service, lock Services, write Service Image Digests, combine Configuration, run A Docker Compose
Command
    #additionalImageTags: # Optional
    #includeSourceTags: false # Optional
    #includeLatestTag: false # Optional
    #buildImages: true # Optional
    #serviceName: # Required when action == Run A Specific Service
    #containerName: # Optional
    #ports: # Optional
    #workingDirectory: # Optional
    #entrypoint: # Optional
    #containerCommand: # Optional
    #detached: true # Optional
    #abortOnContainerExit: true # Optional
    #imageDigestComposeFile: '$(Build.StagingDirectory)/docker-compose.images.yml' # Required when action ==
Write Service Image Digests
    #removeBuildOptions: false # Optional
    #baseResolveDirectory: # Optional
    #outputDockerComposeFile: '$(Build.StagingDirectory)/docker-compose.yml' # Required when action == Lock
Services || Action == Combine Configuration
    #dockerComposeCommand: # Required when action == Run A Docker Compose Command
    #dockerHostEndpoint: # Optional
    #nopIfNoDockerComposeFile: false # Optional
    #requireAdditionalDockerComposeFiles: false # Optional
    #currentWorkingDirectory: '$(System.DefaultWorkingDirectory)' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.

ARGUMENT	DESCRIPTION
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Docker Compose File	(Required) Path to the primary Docker Compose file to use.
Additional Docker Compose Files	(Optional) Additional Docker Compose files to be combined with the primary Docker Compose file. Relative paths are resolved relative to the directory containing the primary Docker Compose file. If a specified file is not found, it is ignored. Specify each file path on a new line.
Environment Variables	(Optional) Environment variables to be set during the command. Specify each name=value pair on a new line.
Project Name	(Optional) Project name used for default naming of images and containers.
Qualify Image Names	(Optional) Qualify image names for built services with the Docker registry connection's hostname if not otherwise specified.
Action	(Required) Select a Docker Compose action.
Additional Image Tags	(Optional) Additional tags for the Docker images being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing Docker images.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing Docker images.
Build Images	(Optional) Build images before starting service containers.
Service Name	(Required) Name of the specific service to run.
Container Name	(Optional) Name of the specific service container to run.
Ports	(Optional) Ports in the specific service container to publish to the host. Specify each host-port:container-port binding on a new line.
Working Directory	(Optional) The working directory for the specific service container.
Entrypoint Override	(Optional) Override the default entry point for the specific service container.

ARGUMENT	DESCRIPTION
Command	(Optional) Command to run in the specific service container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the service containers in the background.
Abort on Container Exit	(Optional) Stop all containers when any container exits.
Image Digest Compose File	(Required) Path to a Docker Compose file that is created and populated with the full image repository digests of each service's Docker image.
Remove Build Options	(Optional) Remove the build options from the output Docker Compose file.
Base Resolve Directory	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
Output Docker Compose File	(Required) Path to an output Docker Compose file.
Command	(Required) Docker Compose command to execute with arguments. For example, 'rm --all' to remove all stopped service containers.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
No-op if no Docker Compose File	(Optional) If the Docker Compose file does not exist, skip this step. This is useful when the step offers optional behavior based on the existence of a Docker Compose file in the repository.
Require Additional Docker Compose Files	(Optional) Produces an error if the additional Docker Compose files do not exist. This overrides the default behavior which is to ignore a file if it does not exist.
Working Directory	(Optional) Working directory for the Docker Compose command.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Build: Go

7/9/2018 • 2 minutes to read • Edit Online

Get, build, test a go application, or run a custom go command.

YAML snippet

```
# Go
# Get, build, or test a Go application, or run a custom Go command.
- task: Go@0
  inputs:
    #command: 'get' # Options: get, build, test, custom
    #customCommand: # Required when command == Custom
    #arguments: # Optional
    workingDirectory:
```

Arguments

ARGUMENT	DESCRIPTION
Command	(Required) Select a Go command to run. Select 'Custom' to use a command not listed here.
Custom command	(Required) Custom Go command for execution. For example: to execute go version, enter version.
Arguments	(Optional) Arguments to the selected command. For example, build time arguments for go build command.
Working Directory	(Required) Current working directory where the script is run. Empty is the root of the repo (build) or artifacts (release), which is $(System.DefaultWorkingDirectory)
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Build: Gradle

7/9/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Build using a Gradle wrapper script

YAML snippet

```
# Gradle
# Build using a Gradle wrapper script
- task: Gradle@2
  inputs:
    #gradleWrapperFile: 'gradlew'
    #workingDirectory: # Optional
    #options: # Optional
    #tasks: 'build'
    #publishJUnitResults: true
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #codeCoverageToolOption: 'None' # Optional. Options: none, cobertura, jaCoCo
    #codeCoverageClassFilesDirectories: 'build/classes/main/' # Required when codeCoverageToolOption == False
    #codeCoverageClassFilter: # Optional
    #codeCoverageFailIfEmpty: false # Optional
    #javaHomeOption: 'JDKVersion' # Options: jDKVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkDirectory: # Required when javaHomeOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
    #gradleOptions: '-Xmx1024m' # Optional
    #sonarQubeRunAnalysis: false
    #sqGradlePluginVersionChoice: 'specify' # Required when sonarQubeRunAnalysis == True# Options: specify,
build
    #sonarQubeGradlePluginVersion: '2.6.1' # Required when sonarQubeRunAnalysis == True &&
SqGradlePluginVersionChoice == Specify
    #checkStyleRunAnalysis: false # Optional
    #findBugsRunAnalysis: false # Optional
    #pmdRunAnalysis: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Gradle Wrapper	The location in the repository of the gradlew wrapper used for the build. For agents on Windows (including Microsoft-hosted agents), you must use the `gradlew.bat` wrapper. Agents on Linux or macOS can use the `gradlew` shell script. See The Gradle Wrapper.
Options	Specify any command line options you want to pass to the Gradle wrapper. See Gradle Command Line.

ARGUMENT	DESCRIPTION
Tasks	The task(s) for Gradle to execute. A list of tasks can be taken from `gradlew tasks` issued from a command prompt. See Gradle Build Script Basics.
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, `**/TEST-*.xml` for all xml files whose name starts with TEST-."
Test Run Title	Assign a title for the JUnit test case results for this build.
CODE COVERAGE	
Code Coverage Tool	Choose a code coverage tool to determine the code that is covered by the test cases for the build.
ADVANCED	
Working Directory	Directory on the build agent where the Gradle wrapper will be invoked from. Defaults to the repository root.
Set JAVA_HOME by JDK Version	Choose which JDK level to run Gradle with. Will attempt to find JDK version and assign JAVA_HOME before running Gradle.
Set JAVA_HOME by Path	Directory on build agent where JDK is located.
JDK Architecture	Optionally supply the architecture (x86, x64) of JDK.
CODE ANALYSIS	
Run SonarQube Analysis	Select if you want to run a SonarQube analysis. See The Gradle build task now supports SonarQube analysis.
Run PMD Analysis	Select if you want to perform a PMD static analysis. A build result page for each project is shown on the **Artifacts** tab of the completed build. See Gradle build task now also supports PMD analysis.
Run Checkstyle Analysis	Select if you want to perform a Checkstyle static analysis The build summary reports the number of issues found by Checkstyle. Detailed issue logs are available under the build Artifact tab of the build summary. If the Checkstyle analysis is customized, the task only attempts to find the reports and produce a summary.
CONTROL OPTIONS	

Example

Build your Java app with Gradle

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

How do I generate a wrapper from my Gradle project?

The Gradle wrapper allows the build agent to download and configure the exact Gradle environment that is checked into the repoistory without having any software configuration on the build agent itself other than the JVM.

1. Create the Gradle wrapper by issuing the following command from the root project directory where your build.gradle resides:

   `jamal@fabrikam> gradle wrapper`

2. Upload your Gradle wrapper to your remote repository.

   There is a binary artifact that is generated by the gradle wrapper (located at `gradle/wrapper/gradle-wrapper.jar`). This binary file is small and doesn't require updating. If you need to change the Gradle configuration run on the build agent, you update the `gradle-wrapper.properties`.

   The repository should look something like this:

```
|-- gradle/
    `-- wrapper/
        `-- gradle-wrapper.jar
        `-- gradle-wrapper.properties
|-- src/
|-- .gitignore
|-- build.gradle
|-- gradlew
|-- gradlew.bat
```

How do I fix timeouts when downloading dependencies?

To fix errors such as `Read timed out` when downloading dependencies, users of Gradle 4.3+ can change the timeout by adding to `Options` `-Dhttp.socketTimeout=60000 -Dhttp.connectionTimeout=60000`. This increases the timeout from 10 seconds to 1 minute.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes 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

YAML snippet

```
# Grunt
# The JavaScript Task Runner
- task: Grunt@0
  inputs:
    #gruntFile: 'gruntfile.js'
    #targets: # Optional
    #arguments: # Optional
    #workingDirectory: # Optional
    #gruntCli: 'node_modules/grunt-cli/bin/grunt'
    #publishJUnitResults: false # Optional
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #enableCodeCoverage: false # Optional
    #testFramework: 'Mocha' # Optional. Options: mocha, jasmine
    #srcFiles: # Optional
    #testFiles: 'test/*.js' # Required when enableCodeCoverage == True
```

Arguments

ARGUMENT	DESCRIPTION
Grunt File Path	Relative path from the repo root to the grunt script that you want to run. The default value is `gruntfile.js`
Grunt task(s)	(Optional) Space delimited list of tasks to run. If you leave it blank, the default task will run.
ADVANCED	
Arguments	Additional arguments passed to grunt. See Using the CLI. Tip: `--gruntfile` is not needed. This argument is handled by the Grunt file path argument shown above.
Working directory	Current working directory when the script is run. If you leave it blank, the working directory is the folder where the script is located.

ARGUMENT	DESCRIPTION
CONTROL OPTIONS	

Example

See Sample Gruntfile.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Node.js streaming task based build system

Demands

gulp

YAML snippet

```
# Gulp
# Node.js streaming task based build system
- task: Gulp@0
  inputs:
    #gulpFile: 'gulpfile.js'
    #targets: # Optional
    #arguments: # Optional
    #workingDirectory: # Optional
    #gulpjs: 'node_modules/gulp/bin/gulp.js'
    #publishJUnitResults: false # Optional
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #enableCodeCoverage: false
    #testFramework: 'Mocha' # Optional. Options: mocha, jasmine
    #srcFiles: # Optional
    #testFiles: 'test/*.js' # Required when enableCodeCoverage == True
```

Arguments

ARGUMENT	DESCRIPTION
Gulp file path	Relative path from the repo root to the gulp script that you want to run. The default value is `gulpfile.js`
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: `--gulpfile` 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.

ARGUMENT	DESCRIPTION
gulp.js location	gulp.js to run. The default value is `node_modules/gulp/bin/gulp.js`
CONTROL OPTIONS	

Example

Run gulp.js

On the Build tab:

Package: npm	Install npm. • Command: `install`
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

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 6 minutes 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 task 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

YAML snippet

```
# Index Sources & Publish Symbols
# Index your source code and publish symbols to a file share or Visual Studio Team Services Symbol Server
- task: PublishSymbols@2
  inputs:
    #symbolsFolder: '$(Build.SourcesDirectory)' # Optional
    #searchPattern: '**/bin/**/*.pdb'
    #indexSources: true # Optional
    #publishSymbols: true # Optional
    #symbolServerType: ' ' # Required when publishSymbols == True# Options:  , teamServices, fileShare
    #symbolsPath: # Optional
    #detailedLog: true # Optional
    #treatNotIndexedAsWarning: false # Optional
    #symbolsMaximumWaitTime: # Optional
    #symbolsProduct: # Optional
    #symbolsVersion: # Optional
    #symbolsArtifactName: 'Symbols_$(BuildConfiguration)' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path to symbols folder	The root path that is searched for symbol files using the search patterns supplied in the next input.
Search pattern	File matching pattern(s) (rooted at the path supplied in the previous input) used to discover `pdbs` that contain symbols.
Index sources	Adds information about the location of the source repository to the symbols. This enables users using these symbols to navigate to the relevant source code.
Publish symbols	Publishes symbols to the symbol server selected in the next inputs.
Symbol server type	**Package Management in Visual Studio Team Services:** • Select this option to use the symbol server built into the Package Management extension. **File share:** • Select this option to use the file share supplied in the next input.
Path to publish symbols	The path to the SymStore file share. To prepare your SymStore symbol store: 1. Set up a folder on a file-sharing server to store the symbols. For example, set up \fabrikam-share\symbols. 2. Grant full control permission to the build agent service account. If you leave this argument blank, your symbols will be source indexed but not published. (You can also store your symbols with your drops. See Publish Build Artifacts).
ADVANCED	
Verbose logging	Enables additional log details.
Warn if not indexed	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. Even if you don't select this option, the messages are written in log.

ARGUMENT	DESCRIPTION
Max wait time (min)	If you want to set a time limit for this task, specify the number of minutes here. The build fails when the limit is reached. If you leave it blank, limit is 2 hours.
Product	If you are publishing your symbols, you can specify the product parameter that is passed to symstore.exe. If blank, $(Build.DefinitionName) is passed.
Version	If you are publishing your symbols, you can specify the version parameter that is passed to symstore.exe. If blank, $(Build.BuildNumber) is passed.
Artifact name	Specify the pattern used for the name of the link from the artifact tab in the build summary to the file share where you are publishing your symbols. For example, if you specify `Symbols_$(BuildConfiguration)`, then the name of the link to your published release symbols would be *Symbols_release*
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 (SRCSRVTRG) where the debugger can find the retrieved source file.

  The mapping information may look something like the following:

```
SRCSRV: variables ----------------------------------------
TFS_EXTRACT_TARGET=%targ%\%var5%\%fnvar%(%var6%)%fnbksl%(%var7%)
TFS_EXTRACT_CMD=tf.exe git view /collection:%fnvar%(%var2%) /teamproject:"%fnvar%(%var3%)"
/repository:"%fnvar%(%var4%)" /commitId:%fnvar%(%var5%) /path:"%var7%" /output:%SRCSRVTRG% %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
SRCSRVVERCTRL=git
SRCSRVERRDESC=access
SRCSRVERRVAR=var2
SRCSRVTRG=%TFS_EXTRACT_TARGET%
SRCSRVCMD=%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_COMMI
T*/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_CO
MMIT*/MyApp/SomeHelper.cs*TFS_APPLY_FILTERS
```

The above example contains two sections: 1) the variables section and 2) the source files section. The information in the variables section is what can be overridden. The variables can leverage other variables, and can leverage information from the source files section.

To override one or more of the variables while debugging with Visual Studio, create an ini file `%LOCALAPPDATA%\SourceServer\srcsrv.ini`. Set the content of the INI file to override the variables. For example:

```
[variables]
TFS_COLLECTION=http://DIFFERENT_SERVER:8080/tfs/DifferentCollection
```

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

How does indexing work?

By choosing to index the sources, an extra section will be injected into the PDB files. PDB files normally contain references to the local source file paths only. For example, `C:\BuildAgent\_work\1\src\MyApp\Program.cs`. The extra section injected into the PDB file contains mapping instructions for debuggers. The mapping information indicates how to retrieve the server item corresponding to each local path.

The Visual Studio debugger will use the mapping information to retrieve the source file from the server. An actual command to retrieve the source file is included in the mapping information. You may be prompted by Visual Studio whether to run the command. For example

```
tf.exe git view /collection:http://SERVER:8080/tfs/DefaultCollection /teamproject:"93fc2e4d-0f0f-4e40-9825-
01326191395d" /repository:"647ed0e6-43d2-4e3d-b8bf-2885476e9c44"
/commitId:3a9910862e22f442cd56ff280b43dd544d1ee8c9 /path:"/MyApp/Program.cs"
/output:"C:\Users\username\AppData\Local\SOURCE~1\TFS_COMMIT\3a991086\MyApp\Program.cs" /applyfilters
```

Can I use source indexing on a portable PDB created from a .NET Core assembly?

No, source indexing is currently not enabled for Portable PDBs as SourceLink doesn't support authenticated source repositories. The workaround at the moment is to configure the build to generate full PDBs. Note that if you are generating a .NET Standard 2.0 assembly and are generating full PDBs and consuming them in a .NET

Framework (full CLR) application then you will be able to fetch sources from 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

7/9/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

Queue a job on a Jenkins server

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Jenkins Queue Job
# Queue a job on a Jenkins server
- task: JenkinsQueueJob@2
  inputs:
    serverEndpoint:
    jobName:
    #isMultibranchJob: # Optional
    #multibranchPipelineBranch: # Required when isMultibranchJob == True
    captureConsole:
    #capturePipeline: # Required when captureConsole == True
    isParameterizedJob:
    #jobParameters: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Jenkins service connection	Select the service connection for your Jenkins instance. To create one, click **Manage** and create a new Jenkins service connection.
Job name	The name of the Jenkins job to queue. This must exactly match the job name on the Jenkins server.
Capture console output and wait for completion	If selected, this task will capture the Jenkins build console output, wait for the Jenkins build to complete, and succeed/fail based on the Jenkins build result. Otherwise, once the Jenkins job is successfully queued, this task will successfully complete without waiting for the Jenkins build to run.

ARGUMENT	DESCRIPTION
Capture pipeline output and wait for pipeline completion	This option is similar to capture console output except it will capture the output for the entire Jenkins pipeline, wait for completion for the entire pipeline, and succeed/fail based on the pipeline result.
Parameterized job	Select this option if the Jekins job requires parameters.
Job parameters	This option is available for parameterized jobs. Specify job parameters, one per line, in the form **parameterName=parameterValue** To set a parameter to an empty value (useful for overriding a default value) leave off the paramter value, e.g. specify **parameterName=** Variables are supported, e.g. to define the **commitId** paramter to be the **git commit ID** for the build, use: **commitId=$(Build.SourceVersion)**. Supported Jenkins parameter types are: • Boolean • String • Choice • Password
Trust server certificate	Selecting this option results in the Jenkins server's SSL certificate being trusted even if it is self-signed or cannot be validated by a Certificate Authority (CA).

Team Foundation Server Plug-in

You can use Team Foundation Server Plug-in (version 5.2.0 or newer) to automatically collect files from the Jenkins workspace and download them into the build.

To set it up:

1. Install the Team Foundation Server Plug-in on the Jenkins server.
2. On the Jenkins server, for each job you would like to collect results from, add the **Collect results for 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).

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes 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

YAML snippet

```
# Maven
# Build with Apache Maven
- task: Maven@2
  inputs:
    #mavenPomFile: 'pom.xml'
    #goals: 'package' # Optional
    #options: # Optional
    #publishJUnitResults: true
    #testResultsFiles: '**/TEST-*.xml' # Required when publishJUnitResults == True
    #testRunTitle: # Optional
    #codeCoverageToolOption: 'None' # Optional. Options: none, cobertura, jaCoCo
    #codeCoverageClassFilter: # Optional
    #codeCoverageClassFilesDirectories: # Optional
    #codeCoverageSourceDirectories: # Optional
    #codeCoverageFailIfEmpty: false # Optional
    #javaHomeOption: 'JDKVersion' # Options: jDKVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkDirectory: # Required when javaHomeOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
    #mavenVersionOption: 'Default' # Options: default, path
    #mavenDirectory: # Required when mavenVersionOption == Path
    #mavenSetM2Home: false # Required when mavenVersionOption == Path
    #mavenOptions: '-Xmx1024m' # Optional
    #mavenAuthenticateFeed: true
    #sonarQubeRunAnalysis: false
    #sqMavenPluginVersionChoice: 'latest' # Required when sonarQubeRunAnalysis == True# Options: latest, pom
    #checkStyleRunAnalysis: false # Optional
    #pmdRunAnalysis: false # Optional
    #findBugsRunAnalysis: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Maven POM file	Relative path from the repo root to the Maven POM .xml file. See Introduction to the POM.

ARGUMENT	DESCRIPTION
Options	Specify any Maven options you want to use.
Goal(s)	In most cases, set this to `package` to compile your code and package it into a .war file. If you leave this argument blank, the build will fail. See Introduction to the Maven build lifecycle.
JUNIT TEST RESULTS	
Publish to 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, `**/TEST-*.xml` for all xml files whose name starts with TEST-."
ADVANCED	
JDK Version	Will attempt to discover the path to the selected JDK version and set JAVA_HOME accordingly.
JDK Architecture	Optionally supply the architecture (x86, x64) of JDK.
CODE ANALYSIS	
Run SonarQube Analysis	Select if you want to run a SonarQube analysis. See The Maven build task now simplifies SonarQube analysis.
Run PMD Analysis	Select if you want to perform a PMD static analysis. A build result page for each .pom file is shown on the **Artifacts** tab of the completed build. See The Maven build task now supports PMD analysis out of the box.
CONTROL OPTIONS	

Example

Build and Deploy your Java application to an Azure web app

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 4 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Build with MSBuild

Demands

msbuild

> **VSTS:** If your team uses Visual Studio 2017 and you want to use our Microsoft-hosted agents, make sure you select as your default queue the **Hosted VS2017**. See Microsoft-hosted agents.

YAML snippet

```
# MSBuild
# Build with MSBuild
- task: MSBuild@1
  inputs:
    #solution: '**/*.sln'
    #msbuildLocationMethod: 'version' # Optional. Options: version, location
    #msbuildVersion: 'latest' # Optional. Options: latest, 15.0, 14.0, 12.0, 4.0
    #msbuildArchitecture: 'x86' # Optional. Options: x86, x64
    #msbuildLocation: # Optional
    #platform: # Optional
    #configuration: # Optional
    #msbuildArguments: # Optional
    #clean: false # Optional
    #maximumCpuCount: false # Optional
    #restoreNugetPackages: false # Optional
    #logProjectEvents: false # Optional
    #createLogFile: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Project	If you want to build a single 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 (`*`) and recursive wildcards (`**`). For example, `**\*.*proj` searches for all MSBuild project (.*proj) files in all subdirectories. Make the sure the projects you specify are downloaded by this build pipeline. On the Repository tab: • If you use TFVC, make sure that the project is a child of one of the mappings on the Repository tab. • If you use Git, make sure that the project or project is in your Git repo, in a branch that you're building. Tip: If you are building a solution, we recommend you use the Visual Studio build task instead of the MSBuild task.
MSBuild Arguments	You can pass additional arguments to MSBuild. For syntax, see MSBuild Command-Line Reference.
Platform	Specify the platform you want to build such as `Win32` , `x86` , `x64` or `any cpu` . Tips: • If you are targeting an MSBuild project (.*proj) file instead of a solution, specify `AnyCPU` (no whitespace). • Declare a build variable such as `BuildPlatform` on the Variables tab (selecting Allow at Queue Time) and reference it here as `$(BuildPlatform)` . This way you can modify the platform when you queue the build and enable building multiple configurations.
Configuration	Specify the configuration you want to build such as `debug` or `release` . Tip: Declare a build variable such as `BuildConfiguration` on the Variables tab (selecting Allow at Queue Time) and reference it here as `$(BuildConfiguration)` . This way you can modify the platform when you queue the build and enable building multiple configurations.
Clean	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. Set to True if you want to rebuild all the code in the code projects. This is equivalent to the MSBuild `/target:clean argument` .
Restore NuGet Packages	**(Important)** This option is deprecated. Make sure to clear this checkbox and instead use the NuGet Installer build task.

ARGUMENT	DESCRIPTION
ADVANCED	
Record Project Details	Select this checkbox if you want details about how much time was needed to build each project. You can see these details when you select this build step in a completed build.
MSBuild	In some cases you might need more control over the version of MSBuild that you are running.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Should I use the Visual Studio Build task or the MSBuild task?

If you are building a solution, in most cases you should use the Visual Studio Build task. This task automatically:

- Sets the `/p:VisualStudioVersion` property for you. This forces MSBuild to use a particular set of targets that increase the likelihood of a successful build.
- Specifies the MSBuild version argument.

In some cases you might need to use the MSBuild task. For example, you should use it if you are building code projects apart from a solution.

Where can I learn more about MSBuild?

MSBuild task

MSBuild reference

MSBuild command-line reference

How do I build multiple configurations for multiple platforms?

1. On the Variables tab, make sure you've got variables defined for your configurations and platforms. To specify multiple values, separate them with commas.

   For example, for a .NET app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	any cpu

   For example, for a C++ app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	x86, x64

2. On the Options tab select **MultiConfiguration** and specify the Multipliers, separated by commas. For example: `BuildConfiguration, BuildPlatform`

   Select Parallel if you want to distribute the jobs (one for each combination of values) to multiple agents in parallel if they are available.

3. On the Build tab, select this step and specify the Platform and Configuration arguments. For example:

   - Platform: `$(BuildPlatform)`
   - Configuration: `$(BuildConfiguration)`

Can I build TFSBuild.proj files?

You cannot build TFSBuild.proj files. These kinds of files are generated by TFS 2005 and 2008. These files contain tasks and targets are supported only using XAML builds.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 Microsoft-hosted agents, select **Hosted VS2017** as your default build queue.. See Microsoft-hosted agents.

YAML snippet

```
# Visual Studio Build
# Build with MSBuild and set the Visual Studio version property.
- task: VSBuild@1
  inputs:
    #solution: '**\*.sln'
    #vsVersion: 'latest' # Optional. Options: latest, 15.0, 14.0, 12.0, 11.0
    #msbuildArgs: # Optional
    #platform: # Optional
    #configuration: # Optional
    #clean: false # Optional
    #maximumCpuCount: false # Optional
    #restoreNugetPackages: false # Optional
    #msbuildArchitecture: 'x86' # Optional. Options: x86, x64
    #logProjectEvents: true # Optional
    #createLogFile: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Solution	If you want to build a single solution, click the **...** button and select the solution. If you want to build multiple solutions, specify search criteria. You can use a single-folder wildcard (`*`) and recursive wildcards (`**`). For example, `**\*.sln` searches for all *.sln* files in all subdirectories. Make the sure the solutions you specify are downloaded by this build pipeline. On the Repository tab: • If you use TFVC, make sure that the solution is a child of one of the mappings on the Repository tab. • If you use Git, make sure that the project or solution is in your Git repo, and in a branch that you're building. Tips: • You can also build MSBuild project (.*proj) files. • If you are building a customized MSBuild project file, we recommend you use the MSBuild task instead of the Visual Studio Build task.
MSBuild Arguments	You can pass additional arguments to MSBuild. For syntax, see MSBuild Command-Line Reference.
Platform	Specify the platform you want to build such as `Win32`, `x86`, `x64` or `any cpu`. Tips: • If you are targeting an MSBuild project (.*proj) file instead of a solution, specify `AnyCPU` (no whitespace). • Declare a build variable such as `BuildPlatform` on the Variables tab (selecting Allow at Queue Time) and reference it here as `$(BuildPlatform)`. This way you can modify the platform when you queue the build and enable building multiple configurations.
Configuration	Specify the configuration you want to build such as `debug` or `release`. Tip: Declare a build variable such as `BuildConfiguration` on the Variables tab (selecting Allow at Queue Time) and reference it here as `$(BuildConfiguration)`. This way you can modify the platform when you queue the build and enable building multiple configurations.

ARGUMENT	DESCRIPTION
Clean	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. Set to True if you want to rebuild all the code in the code projects. This is equivalent to the MSBuild `/target:clean argument`.
Restore NuGet Packages	**(Important)** This option is deprecated. Make sure to clear this checkbox and instead use the NuGet Installer build task.
Visual Studio Version	To avoid problems overall, you must make sure this value matches the version of Visual Studio used to create your solution. The value you select here adds the `/p:VisualStudioVersion={numeric_visual_studio_version}` argument to the MSBuild command run by the build. For example, if you select **Visual Studio 2015**, `/p:VisualStudioVersion=14.0` is added to the MSBuild command. > **VSTS:**If your team wants to use Visual Studio 2017 with our Microsoft-hosted agents, select **Hosted VS2017** as your default build queue. See Microsoft-hosted agents.
ADVANCED	
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.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Should I use the Visual Studio Build task or the MSBuild task?

If you are building a solution, in most cases you should use the Visual Studio Build task. This task automatically:

- Sets the `/p:VisualStudioVersion` property for you. This forces MSBuild to use a particular set of targets

that increase the likelihood of a successful build.

- Specifies the MSBuild version argument.

In some cases you might need to use the MSBuild task. For example, you should use it if you are building code projects apart from a solution.

Where can I learn more about MSBuild?

MSBuild task

MSBuild reference

MSBuild command-line reference

How do I build multiple configurations for multiple platforms?

1. On the Variables tab, make sure you've got variables defined for your configurations and platforms. To specify multiple values, separate them with commas.

   For example, for a .NET app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	any cpu

   For example, for a C++ app you could specify:

Name	Value
BuildConfiguration	debug, release
BuildPlatform	x86, x64

2. On the Options tab select **MultiConfiguration** and specify the Multipliers, separated by commas. For example: `BuildConfiguration, BuildPlatform`

   Select Parallel if you want to distribute the jobs (one for each combination of values) to multiple agents in parallel if they are available.

3. On the Build tab, select this step and specify the Platform and Configuration arguments. For example:

   - Platform: `$(BuildPlatform)`
   - Configuration: `$(BuildConfiguration)`

Can I build TFSBuild.proj files?

You cannot build TFSBuild.proj files. These kinds of files are generated by TFS 2005 and 2008. These files contain tasks and targets are supported only using XAML builds.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Build an Android app with Xamarin

Demands

AndroidSDK, MSBuild, Xamarin.Android

YAML snippet

```
# Xamarin.Android
# Build an Android app with Xamarin
- task: XamarinAndroid@1
  inputs:
    #projectFile: '**/*.csproj'
    #target: # Optional
    #outputDirectory: # Optional
    #configuration: # Optional
    #createAppPackage: true # Optional
    #clean: false # Optional
    #msbuildLocationOption: 'version' # Optional. Options: version, location
    #msbuildVersionOption: '15.0' # Optional. Options: latest, 15.0, 14.0, 12.0, 4.0
    #msbuildFile: # Required when msbuildLocationOption == Location
    #msbuildArchitectureOption: 'x86' # Optional. Options: x86, x64
    #msbuildArguments: # Optional
    #jdkOption: 'JDKVersion' # Options: jDKVersion, path
    #jdkVersionOption: 'default' # Optional. Options: default, 1.10, 1.9, 1.8, 1.7, 1.6
    #jdkDirectory: # Required when jdkOption == Path
    #jdkArchitectureOption: 'x64' # Optional. Options: x86, x64
```

Arguments

ARGUMENT	DESCRIPTION
Project	If you want to build a single Xamarin.Android project, click the **...** button and select the project. If you want to build multiple projects, specify search criteria. You can use a single-folder wildcard (`*`) and recursive wildcards (`**`). For example, `**/*.Android.csproj` searches for all Android.csproj files in all subdirectories in your repo. Note: The projects must have a PackageForAndroid target.
Target	(Optional) Specify the project targets you want to build. Use a semicolon to separate multiple targets.

ARGUMENT	DESCRIPTION
Output Directory	Use a variable to specify the folder where you want the output files to go. For example: `$(build.binariesdirectory)/$(BuildConfiguration)`
Configuration	Specify the configuration you want to build such as `debug` or `release`. Tip: Declare a build variable such as `BuildConfiguration` on the Variables tab (selecting Allow at Queue Time) and reference it here as `$(BuildConfiguration)`. This way you can modify the platform when you queue the build and enable building multiple configurations.
MSBUILD OPTIONS	
MSBuild Location	(Optional) Path to MSBuild (on Windows) or xbuild (on macOS). Default behavior is to search for the latest version.
Additional Arguments	You can pass additional arguments to MSBuild (on Windows) or xbuild (on macOS). For syntax, see MSBuild Command-Line Reference.
JDK OPTIONS	
Select JDK to use for the build	Pick the JDK to be used during the build by selecting a JDK version that will be discovered during builds or by manually entering a JDK path. • JDK Version: Select the JDK version you want to use. • JDK Path: Specify the path to the JDK you want to use.
JDK Architecture	Select x86 or x64.
CONTROL OPTIONS	

Example

Build your Xamarin app

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Build an iOS app with Xamarin on macOS

Demands

Xamarin.iOS

YAML snippet

```
# Xamarin.iOS
# Build an iOS app with Xamarin on macOS.
- task: XamariniOS@2
  inputs:
    #solutionFile: '**/*.sln'
    #configuration: 'Release'
    #clean: false # Optional
    packageApp:
    #buildForSimulator: false # Optional
    #runNugetRestore: false
    #args: # Optional
    #workingDirectory: # Optional
    #mdtoolFile: # Optional
    #signingIdentity: # Optional
    #signingProvisioningProfileID: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Solution	Click the **...** button and select your solution.
Configuration	Specify a configuration such as `Ad-Hoc`, `AppStore`, `Debug`, or `Release`
Create App Package	Select if you want to create an .IPA app package file.
Build for iOS Simulator	Select if you want to build for the iOS Simulator instead of for physical iOS devices.
(OPTIONAL) SIGNING & PROVISIONING	

ARGUMENT	DESCRIPTION
Override Using (Optional)	If the build should use a signing or provisioning method that is different than the default, choose that method here. Choose **File Contents** to use a P12 certificate and provisioning profile. Choose **Identifiers** to retrieve signing settings from the default Keychain and pre-installed profiles. 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. > **Important:** Use a secret variable 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.
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. > **Important:** Select only if you are running one agent per user.
ADVANCED	
Arguments	(Optional) Specify additional command-line arguments for this build.
Working Directory	Working directory for the build. If you leave it blank, it is the root of the repo.
Xbuild Path	(Optional) Specify the path to xbuild. If you leave it blank, the default xbuild path is used.
CONTROL OPTIONS	

Example

Build your Xamarin app

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 6 minutes 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

YAML snippet

```
# Xcode
# Build, test, or archive an Xcode workspace on macOS. Optionally package an app.
- task: Xcode@5
  inputs:
    #actions: 'build'
    #configuration: '$(Configuration)' # Optional
    #sdk: '$(SDK)' # Optional
    #xcWorkspacePath: '**/*.xcodeproj/project.xcworkspace' # Optional
    #scheme: # Optional
    #xcodeVersion: 'default' # Optional. Options: 8, 9, default, specifyPath
    #xcodeDeveloperDir: # Optional
    packageApp:
    #archivePath: # Optional
    #exportPath: 'output/$(SDK)/$(Configuration)' # Optional
    #exportOptions: 'auto' # Optional. Options: auto, plist, specify
    #exportMethod: 'development' # Required when exportOptions == Specify
    #exportTeamId: # Optional
    #exportOptionsPlist: # Required when exportOptions == Plist
    #exportArgs: # Optional
    #signingOption: 'nosign' # Optional. Options: nosign, default, manual, auto
    #signingIdentity: # Optional
    #provisioningProfileUuid: # Optional
    #provisioningProfileName: # Optional
    #teamId: # Optional
    #destinationPlatformOption: 'default' # Optional. Options: default, iOS, tvOS, macOS, custom
    #destinationPlatform: # Optional
    #destinationTypeOption: 'simulators' # Optional. Options: simulators, devices
    #destinationSimulators: 'iPhone 7' # Optional
    #destinationDevices: # Optional
    #args: # Optional
    #workingDirectory: # Optional
    #useXcpretty: # Optional
    #publishJUnitResults: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Actions	Enter a space-delimited list of actions. Valid options are `build`, `clean`, `test`, `analyze`, and `archive`. For example, `clean build` will run a clean build. See the xcodebuild man page.
Configuration	Enter the Xcode project or workspace configuration to be built. The default value of this field is the variable `$(Configuration)`. When using a variable, make sure to specify a value (for example, `Release`) on the Variables tab.

ARGUMENT	DESCRIPTION
SDK	Specify an SDK to use when building the Xcode project or workspace. From the macOS Terminal application, run `xcodebuild -showsdks` to display the valid list of SDKs. The default value of this field is the variable `$(SDK)`. When using a variable, make sure to specify a value (for example, `iphonesimulator`) on the Variables tab.
Workspace or project path	(Optional) Enter a relative path from the root of the repository to the Xcode workspace or project. For example, `MyApp/MyApp.xcworkspace` or `MyApp/MyApp.xcodeproj`.
Scheme	(Optional) Enter a scheme name defined in Xcode. It must be a shared scheme, with its **Shared** checkbox enabled under **Managed Schemes** in Xcode. If you specify a **Workspace or project path** above without specifying a scheme, and the workspace has a single shared scheme, it will be automatically used.
Xcode version	Specify the target version of Xcode. Select `Default` to use the default version of Xcode on the agent machine. Selecting a version number (e.g. `Xcode 9`) relies on environment variables being set on the agent machine for the version's location (e.g. `XCODE_9_DEVELOPER_DIR=/Applications/Xcode_9.0.0.app/Contents/Devel`). Select `Specify path` 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. `/Applications/Xcode_9.0.0.app/Contents/Developer`). This is useful when multiple versions of Xcode are installed on the agent machine.
(OPTIONAL) SIGNING & PROVISIONING	
Signing style	Choose the method of signing the build. Select `Do not code sign` to disable signing. Select `Project defaults` to use only the project's signing configuration. Select `Manual signing` to force manual signing and optionally specify a signing identity and provisioning profile. Select `Automatic signing` to force automatic signing and optionally specify a development team ID. If your project requires signing, use the "Install Apple..." tasks to install certificates and provisioning profiles prior to the Xcode build.
Signing identity	(Optional) Enter a signing identity override with which to sign the build. This may require unlocking the default keychain on the agent machine. If no value is entered, the Xcode project's setting will be used.
Provisioning profile UUID	(Optional) Enter the UUID of an installed provisioning profile to be used for this build. Use separate build tasks with different schemes or targets to specify separate provisioning profiles by target in a single workspace (iOS, tvOS, watchOS).
Team ID	(Optional, unless you are a member of multiple development teams.) Specify the 10-character development team ID.
PACKAGE OPTIONS	
Create app package	Indicate whether an IPA app package file should be generated as a part of the build.
Archive path	(Optional) Specify a directory where created archives should be placed.

ARGUMENT	DESCRIPTION
Export path	(Optional) Specify the destination for the product exported from the archive.
Export options	Select a way of providing options for exporting the archive. When the default value of `Automatic` is selected, the export method is automatically detected from the archive. Select `Plist` to specify a plist file containing export options. Select `Specify` to provide a specific **Export method** and **Team ID**.
Export method	Enter the method that Xcode should use to export the archive. For example: `app-store`, `package`, `ad-hoc`, `enterprise`, or `development`.
Team ID	(Optional) Enter the 10-character team ID from the Apple Developer Portal to use during export.
Export options plist	Enter the path to the plist file that contains options to use during export.
Export arguments	(Optional) Enter additional command line arguments to be used during export.
DEVICES & SIMULATORS	
Destination platform	Select the destination device's platform to be used for UI testing when the generic build device isn't valid. Choose `Custom` to specify a platform not included in this list. When `Default` is selected, no simulators nor devices will be targeted.
Destination type	Choose the destination type to be used for UI testing. Devices must be connected to the Mac performing the build via a cable or network connection. See **Devices and Simulators** in Xcode.
Simulators	Enter an Xcode simulator name to be used for UI testing. For example, enter `iPhone X` (iOS and watchOS) or `Apple TV 4K` (tvOS). A target OS version is optional and can be specified in the format 'OS=*versionNumber*', such as `iPhone X,OS=11.1`. A list of simulators installed on the **Hosted macOS** agent can be found here.
Devices	Enter the name of the device to be used for UI testing, such as `Raisa's iPad`. Only one device is currently supported. Note that Apple does not allow apostrophes (`'`) in device names. Instead, right single quotation marks (`'`) can be used.
ADVANCED	
Arguments	(Optional) Enter additional command line arguments with which to build. This is useful for specifying `-target` or `-project` arguments instead of specifying a workspace/project and scheme. See the xcodebuild man page.
Working directory	(Optional) Enter the working directory in which to run the build. If no value is entered, the root of the repository will be used.
Output directory	Enter a path relative to the working directory where build output (binaries) will be placed. The default value includes variables. When these are used, make sure to specify values on the Variables tab.

ARGUMENT	DESCRIPTION
Use xcpretty	Specify whether to use xcpretty to format xcodebuild output and generate JUnit test results. Enabling this requires xcpretty to be installed on the agent machine. It is preinstalled on VSTS hosted build agents. See xcpretty on GitHub.
Publish test results to VSTS/TFS	If xcpretty is enabled above, specify whether to publish JUnit test results to VSTS/TFS.
CONTROL OPTIONS	

Example

Build your Xcode app

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Build: Xcode Package iOS

7/9/2018 • 2 minutes 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

YAML snippet

```
# Xcode Package iOS
# Generate an .ipa file from Xcode build output using xcrun (Xcode 7 or below)
- task: XcodePackageiOS@0
  inputs:
    #appName: 'name.app'
    #ipaName: 'name.ipa'
    provisioningProfile:
    #sdk: 'iphoneos'
    #appPath: '$(SDK)/$(Configuration)/build.sym/$(Configuration)-$(SDK)'
    #ipaPath: '$(SDK)/$(Configuration)/build.sym/$(Configuration)-$(SDK)/output'
```

Arguments

ARGUMENT	DESCRIPTION
Name of .app	Name of the .app file, which is sometimes different from the .ipa file.
Name of .ipa	Name of the .ipa file, which is sometimes different from the .app file.
Provisioning Profile Name	Name of the provisioning profile to use when signing.
SDK	The SDK you want to use. Run **xcodebuild -showsdks** to see a list of valid SDK values.
ADVANCED	
Path to .app	Relative path to the built .app file. The default value is `$(SDK)/$(Configuration)/build.sym/$(Configuration)-$(SDK)`. Make sure to specify the variable values on the variables tab.

ARGUMENT	DESCRIPTION
Path to place .ipa	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 variables tab.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 3 minutes 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

YAML snippet

```
# Archive Files
# Archive files using compression formats such as .7z, .rar, .tar.gz, and .zip.
- task: ArchiveFiles@2
  inputs:
    #rootFolderOrFile: '$(Build.BinariesDirectory)'
    includeRootFolder:
    #archiveType: 'zip' # Options: zip, 7z, tar, wim
    #tarCompression: 'gz' # Optional. Options: gz, bz2, xz, none
    #archiveFile: '$(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip'
    #replaceExistingArchive: true
```

Arguments

ARGUMENT	DESCRIPTION
Root folder (or file) to archive	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 `$(Build.SourcesDirectory)`). If the specified path is a folder, recursively, all nested files and folders will be included in the archive. If the specified path is a file, only the single file will be included in the archive.
Prefix root folder name to archive paths	If selected, the root folder name will be prefixed to file paths within the archive. Otherwise, all file paths will start one level lower. For example, suppose the selected root folder is: `/home/user/output/classes/`, and contains: `com/acme/Main.class`. • If selected, the resulting archive would contain: `classes/com/acme/Main.class`. • Otherwise, the resulting archive would contain: `com/acme/Main.class`..

ARGUMENT	DESCRIPTION
Archive type	Specify the compression scheme used. To create `foo.jar`, for example, choose `zip` for the compression, and specify `foo.jar` as the archive file to create. For all tar files (including compressed ones), choose `tar`. • `zip` - default, zip format, choose this for all zip compatible types, (.zip, .jar, .war, .ear) • `7z` - 7-Zip format, (.7z) • `tar` - tar format, choose this for compressed tars, (.tar.gz, .tar.bz2, .tar.xz) • `wim` - wim format, (.wim)
Tar compression	Only applicable if the `tar` archive type is selected. Optionally choose a compression scheme, or choose `None` to create an uncompressed tar file. • `gz` - default, gzip compression (.tar.gz, .tar.tgz, .taz) • `bz2` - bzip2 compression (.tar.bz2, .tz2, .tbz2) • `xz` - xz compression (.tar.xz, .txz) • `None` - no compression, choose this to create a uncompressed tar file (.tar)
Archive file to create	Specify the name of the archive file to create. The file extension should match the selected archive type. For example to create `foo.tgz`, select the `tar` archive type, `gz` for tar compression.
Replace existing archive	If an existing archive exists, specify whether to overwrite it. Otherwise, files will be added to it as long as it is not a comprssed tar. If adding to an existing archive, these types are supported: • `zip` • `7z` • `tar` - uncompressed only • `wim`
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.

2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 Network Load Balancer

7/9/2018 • 2 minutes to read • Edit Online

Connect/Disconnect an Azure virtual machine's network interface to a Load Balancer's backend address pool

YAML snippet

```
# Azure Network Load Balancer
# Connect/Disconnect an Azure virtual machine's network interface to a Load Balancer's backend address pool
- task: AzureNLBManagement@1
  inputs:
    azureSubscription:
    resourceGroupName:
    loadBalancer:
    action: # Options: disconnect, connect
```

Arguments

ARGUMENT	DESCRIPTION
Azure Subscription	(Required) Select the Azure Resource Manager subscription for the deployment.
Resource Group	(Required) Select the resource group name.
Load Balancer Name	(Required) Select or enter the load balancer.
Action	(Required) Disconnect: Removes the virtual machine's primary network interface from the load balancer's backend pool. So that it stops receiving network traffic. Connect: Adds the virtual machine's primary network interface to load balancer backend pool. So that it starts receiving network traffic based on the load balancing rules for the load balancer resource.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Bash

7/9/2018 • 2 minutes to read • Edit Online

This is an early preview. Run a Bash script on macOS, Linux, or Windows

YAML snippet

```
# Bash
# Run a Bash script on macOS, Linux, or Windows
- task: Bash@3
  inputs:
    #targetType: 'filePath' # Optional. Options: filePath, inline
    #filePath: # Required when targetType == FilePath
    #arguments: # Optional
    #script: '# Write your commands here# Use the environment variables input below to pass secret variables
to this script' # Required when targetType == Inline
    #workingDirectory: # Optional
    #failOnStderr: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Type	(Optional) Target script type: File Path or Inline
Script Path	(Required) Path of the script to execute. Must be a fully qualified path or relative to $(System.DefaultWorkingDirectory).
Arguments	(Optional) Arguments passed to the PowerShell script. Either ordinal parameters or named parameters.
Script	(Required)
Working Directory	(Optional) undefined
Fail on Standard Error	(Optional) If this is true, this task will fail if any errors are written to the StandardError stream.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Batch script

7/9/2018 • 2 minutes 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.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# Batch Script
# Run a windows cmd or bat script and optionally allow it to change the environment
- task: BatchScript@1
  inputs:
    filename:
    #arguments: # Optional
    #modifyEnvironment: False # Optional
    #workingFolder: # Optional
    #failOnStandardError: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path	Specify the path to the .bat or .cmd script you want to run. The path must be a fully qualified path or a valid path relative to the default working directory. In Team Foundation Build, this directory is $(Build.SourcesDirectory).
Arguments	Specify arguments to pass to the script.
Modify environment	Select this check box if you want 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 pipeline, add this task:

Utility: Batch Script	Run test.bat. • Path: `test.bat`

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn 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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Run a program from the command prompt.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Command Line
# Run a command line script using cmd.exe on Windows and bash on macOS and Linux.
- task: CmdLine@2
  inputs:
    #script: 'echo Write your commands hereecho Use the environment variables input below to pass secret
variables to this script'
    #workingDirectory: # Optional
    #failOnStderr: false # Optional
```

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 enviornment 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 $(Build.SourcesDirectory).
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 pipeline, add these tasks:

Utility: Command Line	Get the date. • Tool: `date` • Arguments: `/t`
Utility: Command Line	Display the operating system version. • Tool: `ver`
Utility: Command Line	Display the environment variables. • Tool: `set`
Utility: Command Line	Display all files in all the folders created by the build pipeline. • Tool: `dir` • Arguments: `/s` • Advanced, Working folder: `$(Agent.BuildDirectory)`

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn Windows commands?

An A-Z Index of the Windows CMD command line

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

5/31/2018 • 2 minutes 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 `$(Build.SourcesDirectory)`). If your build produces artifacts outside of the sources directory, specify `$(Agent.BuildDirectory)` 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: • `**` copies all files in the root folder. • `**\*` copies all files in the root folder and all files in all sub-folders. • `**\bin` copies files in any sub-folder named bin.
Artifact Name	Specify the name of the artifact. For example: `drop`
Artifact Type	Choose **server** to store the artifact on your Team Foundation Server. This is the best and simplest option in most cases. See Artifacts in Team Build.
CONTROL OPTIONS	

Q & A

Q: This step didn't produce the outcome I was expecting. How can I fix it?

This task has a couple of known issues:

- Some minimatch patterns don't work.
- It eliminates the most common root path for all paths matched.

You can avoid these issues by instead using the Copy Files task and the Publish Build Artifacts task.

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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

7/9/2018 • 3 minutes 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.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Copy Files
# Copy files from source folder to target folder using match patterns (The match patterns will only match
file paths, not folder paths)
- task: CopyFiles@2
  inputs:
    #sourceFolder: # Optional
    #contents: '**'
    targetFolder:
    #cleanTargetFolder: false # Optional
    #overWrite: false # Optional
    #flattenFolders: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Source Folder	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 `$(Build.SourcesDirectory)`). If your build produces artifacts outside of the sources directory, specify `$(Agent.BuildDirectory)` to copy files from the directory created for the pipeline.

ARGUMENT	DESCRIPTION
Contents	Specify match pattern filters (one on each line) that you want to apply to the list of files to be copied. For example: • `*` copies all files in the root folder. • `**\*` copies all files in the root folder and all files in all sub-folders. • `**\bin\**` copies all files recursively from any `bin` folder. The pattern is used to match only file paths, not folder paths. So you should specify patterns such as `**\bin\**` instead of `**\bin`. More examples are shown below.
Target Folder	Folder where the files will be copied. In most cases you specify this folder using a variable. For example, specify `$(Build.ArtifactStagingDirectory)` if you intend to publish the files as build artifacts.
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.
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
```

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

File matching patterns reference

How do I use this task to publish artifacts?

See Artifacts in 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

7/9/2018 • 2 minutes 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

YAML snippet

```
# cURL Upload Files
# Use cURL to upload files with FTP, FTPS, SFTP, HTTP, and more.
- task: cURLUploader@2
  inputs:
    files:
    #authType: 'ServiceEndpoint' # Optional. Options: serviceEndpoint, userAndPass
    #serviceEndpoint: # Required when authType == ServiceEndpoint
    #username: # Optional
    #password: # Optional
    #url: # Required when authType == UserAndPass
    #remotePath: 'upload/$(Build.BuildId)/' # Optional
    #options: # Optional
    #redirectStderr: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Files	If you want to upload a single file, click the **...** button and select the file. If you want to upload multiple files, specify a minimatch pattern filter. For example, specify `**\*.zip` to upload all ZIP files in all sub-folders.
Username	Specify the username for server authentication.
Password	Specify the password for server authentication. > **Important:** Use a secret variable to avoid exposing this value.

ARGUMENT	DESCRIPTION
URL	URL to the location where you want to upload the files. If you are uploading to a folder, make sure to end the argument with a trailing slash. Acceptable URL protocols include `DICT:// `, `FILE:// `, `FTP:// `, `FTPS:// `, `GOPHER:// `, `HTTP:// `, `HTTPS:// `, `IMAP:// `, `IMAPS:// `, `LDAP:// `, `LDAPS:// `, `POP3:// `, `POP3S:// `, `RTMP:// `, `RTSP:// `, `SCP:// `, `SFTP:// `, `SMTP:// `, `SMTPS:// `, `TELNET:// `, and `TFTP:// `.
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 **--stderr -** 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.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

File matching patterns reference

Where can I learn FTP commands?

List of raw FTP commands

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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: Decrypt File (OpenSSL)

7/9/2018 • 2 minutes to read • Edit Online

A thin utility task for file decryption using OpenSSL.

YAML snippet

```
# Decrypt File (OpenSSL)
# A thin utility task for file decryption using OpenSSL.
- task: DecryptFile@1
  inputs:
    #cipher: 'des3'
    inFile:
    passphrase:
    #outFile: # Optional
    #workingDirectory: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Cypher	(Required) Encryption cypher to use. See cypher suite names for a complete list of possible values.
Encrypted file	(Required) Relative path of file to decrypt.
Passphrase	(Required) Passphrase to use for decryption. **Use a Variable to encrypt the passphrase.**
Decrypted file path	(Optional) Optional filename for decrypted file. Defaults to the Encrypted File with a ".out" extension
Working directory	(Optional) Working directory for decryption. Defaults to the root of the repository.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Delay

7/9/2018 • 2 minutes to read • Edit Online

Pause execution of the process for a fixed delay time.

Demands

Can be used in only an agentless phase of a release pipeline.

YAML snippet

```
# Delay
# Delay further execution of the workflow by a fixed time.
- task: Delay@1
  inputs:
    #delayForMinutes: '0'
```

Arguments

PARAMETER	COMMENTS
Display name	Required. The name to display for this task.
Delay Time (minutes)	Required. The number of minutes to delay execution.
Control options	See Control options

Also see this task on GitHub.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Utility: Delete files

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015.3

Delete files or folders.

Demands

None

YAML snippet

```
# Delete Files
# Delete files or folders. (The minimatch patterns will only match file paths, not folder paths)
- task: DeleteFiles@1
  inputs:
    #sourceFolder: # Optional
    #contents: 'myFileShare'
```

Arguments

ARGUMENT	DESCRIPTION
Source Folder	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 `$(Build.SourcesDirectory)`). If your build produces artifacts outside of the sources directory, specify `$(Agent.BuildDirectory)` to delete files from the build agent working directory.
Contents	Specify minimatch pattern filters (one on each line) that you want to apply to the list of files to be deleted. For example: • `**` deletes all files and folders in the root folder. • `temp` deletes the temp folder in the root folder. • `temp*` deletes any file or folder in the root folder with a name that begins with temp. • `**\temp\**` deletes all files in any sub-folder named temp. • `**\temp*` deletes any file or folder with a name that begins with temp. • `**\temp*\**` deletes files in any sub-folder that begins with the name temp.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Q: What's a minimatch pattern? How does it work?

A: See:

- https://github.com/isaacs/minimatch
- https://realguess.net/tags/minimatch/

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

Download a secure file to a temporary location on the build or release agent

Use this task to download a secure file from the server during a build or release.

Once downloaded, the secure file is located in the `$(Agent.TempDirectory)` directory of the VSTS Agent.

The full path of the downloaded file is stored to the `$env:DOWNLOADSECUREFILE_SECUREFILEPATH` environment variable.

If you use multiple versions of the Download Secure File task in your pipeline, they can be referenced with the `$env:DOWNLOADSECUREFILE1_SECUREFILEPATH`, `$env:DOWNLOADSECUREFILE2_SECUREFILEPATH`, `...` environment variables, where the number in the environment variable corresponds with the task version.

Note that if you use two Download Secure File tasks in the same pipeline with the same task version, the `$env:DOWNLOADSECUREFILE_SECUREFILEPATH` environment variable will not be populated, but both files will still be downloaded to `$(Agent.TempDirectory)`.

YAML snippet

```
# Download Secure File
# Download a secure file to a temporary location on the build or release agent
- task: DownloadSecureFile@1
  inputs:
    secureFile:
```

Arguments

ARGUMENT	DESCRIPTION
Secure File	Select the secure file to download to a temporary location on the agent machine. The file will be deleted after the build or release.

Utility: Download Build Artifacts

7/9/2018 • 2 minutes to read • Edit Online

Download Build Artifacts

YAML snippet

```
# Download Build Artifacts
# Download Build Artifacts
- task: DownloadBuildArtifacts@0
  inputs:
    #buildType: 'current' # Options: current, specific
    #project: # Required when buildType == Specific
    #pipeline: # Required when buildType == Specific
    #specificBuildWithTriggering: false # Optional
    #buildVersionToDownload: 'latest' # Required when buildType == Specific# Options: latest,
latestFromBranch, specific
    #branchName: 'refs/heads/master' # Required when buildType == Specific && BuildVersionToDownload ==
LatestFromBranch
    #buildId: # Required when buildType == Specific && BuildVersionToDownload == Specific
    #downloadType: 'single' # Options: single, specific
    #artifactName: # Required when downloadType == Single
    #itemPattern: '**' # Optional
    #downloadPath: '$(System.ArtifactsDirectory)'
    #parallelizationLimit: '8' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Download artifacts produced by	(Required) Download artifacts produced by the current build, or from a specific build.
Project	(Required) The project from which to download the build artifacts
Build pipeline	(Required) Select the build pipeline name
When appropriate, download artifacts from the triggering build.	(Optional) If checked, this build task will try to download artifacts from the triggering build. If there is no triggering build from the specified pipeline, it will download artifacts from the build specified in the options below.
Build version to download	(Required) undefined
Branch name	(Required) Specify to filter on branch/ref name, for example: `refs/heads/develop`.
Build	(Required) The build from which to download the artifacts
Download type	(Required) Download a specific artifact or specific files from the build.

ARGUMENT	DESCRIPTION
Artifact name	(Required) The name of the artifact to download
Matching pattern	(Optional) Specify files to be downloaded as multi line minimatch pattern. More Information The default pattern (**) will download all files across all artifacts in the build. To download all files within artifact drop use drop/**.
Destination directory	(Required) Path on the agent machine where the artifacts will be downloaded
Parallelization limit	(Optional) Number of files to download simultaneously
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Download Package

7/9/2018 • 2 minutes to read • Edit Online

Download a package from a Package Management feed in VSTS or TFS. Requires the Package Management extension.

YAML snippet

```
# Download Package
# Download a package from a Package Management feed in VSTS or TFS.  Requires the Package Management
extension.
- task: DownloadPackage@0
  inputs:
    feed:
    pipeline:
    version:
    #downloadPath: '$(System.ArtifactsDirectory)'
```

Arguments

ARGUMENT	DESCRIPTION
Feed	(Required) Select the package source
Package	(Required) Select the package to download, only nuget packages are supported currently
Version	(Required) Version of the package
Destination directory	(Required) Path on the agent machine where the package will be downloaded
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Extract Files

7/9/2018 • 2 minutes 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

YAML snippet

```
# Extract Files
# Extract a variety of archive and compression files such as .7z, .rar, .tar.gz, and .zip.
- task: ExtractFiles@1
  inputs:
    #archiveFilePatterns: '*.zip'
    destinationFolder:
    #cleanDestinationFolder: true
```

Arguments

ARGUMENT	DESCRIPTION
Archive file patterns	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 `$(Build.SourcesDirectory)`). Specify match pattern filters (one on each line) that you want to apply to identify the list of archives to extract. For example: • `test.zip` extracts the test.zip file to the root folder. • `test\*.zip` extracts all .zip files in the test folder. • `**\*.tar` extracts all .tar files in the root folder and sub-folders. • `**\bin\*.7z` extracts all ".7z" files in any sub-folder named bin. The pattern is used to match only archive file paths, not folder paths, and not archive contents to be extracted. So you should specify patterns such as `**\bin\**` instead of `**\bin`.
Destination folder	Folder where the archives will be extracted. The default file path is relative to the root folder of the repo (same as if you had specified `$(Build.SourcesDirectory)`).
Clean destination folder before extracting	Select this check box to delete all existing files in the destination folder before beginning to extract archives.

ARGUMENT	DESCRIPTION
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

File matching patterns reference

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes 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.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# FTP Upload
# FTP Upload
- task: FtpUpload@1
  inputs:
    #credentialsOption: 'serviceEndpoint' # Options: serviceEndpoint, inputs
    #serverEndpoint: # Required when credentialsOption == ServiceEndpoint
    #serverUrl: # Required when credentialsOption == Inputs
    #username: # Required when credentialsOption == Inputs
    #password: # Required when credentialsOption == Inputs
    rootDirectory:
    #filePatterns: '**'
    #remoteDirectory: '/upload/$(Build.BuildId)/'
    #clean: false
    #cleanContents: false # Required when clean == False
    #overwrite: true
    #preservePaths: false
    #trustSSL: false
```

Arguments

ARGUMENT	DESCRIPTION
FTP service connection	Select the service connection for your FTP server. To create one, click the Manage link and create a new Generic service connection, enter the FTP server URL for the server URL, e.g. `ftp://server.example.com`, and required credentials. Secure connections will always be made regardless of the specified protocol (`ftp://` or `ftps://`) if the target server supports FTPS. To allow only secure connections, use the `ftps://` protocol, e.g. `ftps://server.example.com`. Connections to servers not supporting FTPS will fail if `ftps://` is specified.

ARGUMENT	DESCRIPTION
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 `$(Build.SourcesDirectory)`).
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 `**`.
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 ftps://, even if it is self-signed or cannot be validated by a Certificate Authority (CA).
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about file matching patterns?

File matching patterns reference

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

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

YAML snippet

```
# Install Apple Certificate
# Install an Apple certificate required to build on a macOS agent
- task: InstallAppleCertificate@2
  inputs:
    certSecureFile:
    #certPwd: # Optional
    #keychain: 'temp' # Options: default, temp, custom
    #keychainPassword: # Required when keychain == Custom || Keychain == Default
    #customKeychainPath: # Required when keychain == Custom
    #deleteCert: # Optional
    #deleteCustomKeychain: # Optional
    #signingIdentity: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Certificate (P12)	Select the certificate (.p12) that was uploaded to **Secure Files** to install on the macOS agent.
Certificate (P12) Password	Password to the Apple certificate (.p12). Use a new build variable with its lock enabled on the **Variables** tab to encrypt this value.
Advanced - Keychain	Select the keychain in which to install the Apple certificate. You can choose to install the certificate in a temporary keychain (default), the default keychain or a custom keychain. A temporary keychain will always be deleted after the build or release is complete.
Advanced - Keychain Password	Password to unlock the keychain. Use a new build variable with its lock enabled on the **Variables** tab to encrypt this value. A password is generated for the temporary keychain if not specified.
Advanced - Delete Certificate from Keychain	Select to delete the certificate from the keychain after the build or release is complete. This option is visible when custom keychain or default keychain are selected.

ARGUMENT	DESCRIPTION
Advanced - Custom Keychain Path	Full path to a custom keychain file. The keychain will be created if it does not exist. This option is visible when a custom keychain is selected.
Advanced - Delete Custom Keychain	Select to delete the custom keychain from the agent after the build or release is complete. This option is visible when a custom keychain is selected.

Utility: Install Apple Provisioning Profile

7/9/2018 • 2 minutes to read • Edit Online

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

YAML snippet

```
# Install Apple Provisioning Profile
# Install an Apple provisioning profile required to build on a macOS agent
- task: InstallAppleProvisioningProfile@1
  inputs:
    #provisioningProfileLocation: 'secureFiles' # Options: secureFiles, sourceRepository
    #provProfileSecureFile: # Required when provisioningProfileLocation == SecureFiles
    #provProfileSourceRepository: # Required when provisioningProfileLocation == SourceRepository
    #removeProfile: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Provisioning Profile Location (**VSTS**)	Select the location of the provisioning profile to install. The provisioning profile can be uploaded to **Secure Files** or stored in your source repository or a local path on the agent.
Provisioning Profile	Select the provisioning profile that was uploaded to **Secure Files** to install on the macOS agent (or) Select the provisioning profile from the source repository or specify the local path to a provisioning profile on the macOS agent.
Remove Profile After Build	Select to specify that the provisioning profile should be removed from the agent after the build or release is complete.

Utility: Install SSH Key

7/9/2018 • 2 minutes to read • Edit Online

Install an SSH key prior to a build or release

YAML snippet

```
# Install SSH Key
# Install an SSH key prior to a build or release
- task: InstallSSHKey@0
  inputs:
    hostName:
    sshPublicKey:
    #sshPassphrase: # Optional
    sshKeySecureFile:
```

Arguments

ARGUMENT	DESCRIPTION
Known Hosts Entry	(Required) The entry for this SSH key for the known_hosts file.
SSH Public Key	(Required) The contents of the public SSH key.
SSH Passphrase	(Optional) The passphrase for the SSH key, if any.
SSH Key	(Required) Select the SSH key that was uploaded to `Secure Files` to install on the agent.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Jenkins Download Artifacts

7/9/2018 • 2 minutes to read • Edit Online

Download artifacts produced by a Jenkins job

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

```
# Jenkins Download Artifacts
# Download artifacts produced by a Jenkins job
- task: JenkinsDownloadArtifacts@1
  inputs:
    jenkinsServerConnection:
    jobName:
    #jenkinsJobType: # Optional
    #saveTo: 'jenkinsArtifacts'
    #jenkinsBuild: 'LastSuccessfulBuild' # Options: lastSuccessfulBuild, buildNumber
    #jenkinsBuildNumber: '1' # Required when jenkinsBuild == BuildNumber
    #itemPattern: '**' # Optional
    #downloadCommitsAndWorkItems: # Optional
    #startJenkinsBuildNumber: # Optional
    #artifactDetailsFileNameSuffix: # Optional
    #propagatedArtifacts: false # Optional
    #artifactProvider: 'azureStorage' # Required when propagatedArtifacts == NotValid# Options: azureStorage
    #connectedServiceNameARM: # Required when propagatedArtifacts == True
    #storageAccountName: # Required when propagatedArtifacts == True
    #containerName: # Required when propagatedArtifacts == True
    #commonVirtualPath: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Jenkins service connection	(Required) Select the service connection for your Jenkins instance. To create one, click the Manage link and create a new Jenkins service connection.
Job name	(Required) The name of the Jenkins job to download artifacts from. This must exactly match the job name on the Jenkins server.
Jenkins job type	(Optional) Jenkins job type, detected automatically.
Save to	(Required) Jenkins artifacts will be downloaded and saved to this directory. This directory will be created if it does not exist.
Download artifacts produced by	(Required) Download artifacts produced by the last successful build, or from a specific build instance.

ARGUMENT	DESCRIPTION
Jenkins build number	(Required) Download artifacts produced by this build.
Item Pattern	(Optional) Specify files to be downloaded as multi line minimatch pattern. More Information The default pattern (**) will download all files across all artifacts produced by the Jenkins job. To download all files within artifact drop use drop/**.
Download Commits and WorkItems	(Optional) Enables downloading the commits and workitem details associated with the Jenkins Job
Download commits and workitems from	(Optional) Optional start build number for downloading commits and work items. If provided, all commits and work items between start build number and build number given as input to download artifacts will be downloaded.
Commit and WorkItem FileName	(Optional) Optional file name suffix for commits and workitem attachment. Attachment will be created with commits_{suffix}.json and workitem_{suffix}.json. If this input is not provided attachments will be create with the name commits.json and workitems.json
Artifacts are propagated to Azure	(Optional) Check this if Jenkins artifacts were propagated to Azure. To upload Jenkins artifacts to azure, refer to this Jenkins plugin
Artifact Provider	(Required) Choose the external storage provider used in Jenkins job to upload the artifacts.
Azure Subscription	(Required) Choose the Azure Resource Manager subscription for the artifacts.
Storage Account Name	(Required) Azure Classic and Resource Manager stoarge accounts are listed. Select the Storage account name in which the artifacts are propagated.
Container Name	(Required) Name of the container in the storage account to which artifacts are uploaded.
Common Virtual Path	(Optional) Path to the artifacts inside the Azure storage container.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Invoke Azure function

7/9/2018 • 2 minutes to read • Edit Online

Invoke a HTTP triggered function in an Azure function app and parse the response.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

Can be used in only an agentless phase of a release pipeline.

YAML snippet

```
# Invoke Azure Function
# Invoke Azure function as a part of your process.
- task: AzureFunction@1
  inputs:
    function:
    key:
    #method: 'POST' # Options: oPTIONS, gET, hEAD, pOST, pUT, dELETE, tRACE, pATCH
    #headers: '{Content-Type:application/json, PlanUrl: $(system.CollectionUri), ProjectId:
$(system.TeamProjectId), HubName: $(system.HostType), PlanId: $(system.PlanId), JobId: $(system.JobId),
TimelineId: $(system.TimelineId), TaskInstanceId: $(system.TaskInstanceId), AuthToken: $(system.AccessToken)}'
    #queryParameters: # Optional
    #body: # Required when method != GET && Method != HEAD
    #waitForCompletion: 'false' # Options: true, false
    #successCriteria: # Optional
```

Arguments

PARAMETER	COMMENTS
Azure function URL	Required. The URL of the Azure function to be invoked.
Function key	Required. The value of the available function or the host key for the function to be invoked. Should be secured by using a hidden variable.
Method	Required. The HTTP method with which the function will be invoked.
Headers	Optional. The header in JSON format to be attached to the request sent to the function.
Query parameters	Optional. Query parameters to append to the function URL. Must not start with "**?**" or "**&**".

PARAMETER	COMMENTS
Body	Optional. The request body for the Azure function call in JSON format.
Completion Event	Required. How the task reports completion. Can be **API response** (the default) - completion is when function returns success and success criteria evaluates to true, or **Callback** - the Azure function makes a callback to update the timeline record.
Success criteria	Optional. How to parse the response body for success.
Control options	See Control options

Succeeds if the function returns success and the response body parsing is successful, or when the function updates the timeline record with success.

For more information about using this task, see Approvals and gates overview.

Also see this task on GitHub.

Utility: Invoke REST API

7/10/2018 • 2 minutes to read • Edit Online

Invoke an HTTP API and parse the response.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This task is available in both builds and releases in TFS 2018.2 In TFS 2018 RTM, this task is available only in releases.

Demands

This task can be used in only an agentless phase.

YAML snippet

```
# Invoke REST API
# Invoke REST API as a part of your process.
- task: InvokeRESTAPI@1
  inputs:
    serviceConnection:
    #method: 'POST' # Options: oPTIONS, gET, hEAD, pOST, pUT, dELETE, tRACE, pATCH
    #headers: '{Content-Type:application/json, PlanUrl: $(system.CollectionUri), ProjectId:
$(system.TeamProjectId), HubName: $(system.HostType), PlanId: $(system.PlanId), JobId: $(system.JobId),
TimelineId: $(system.TimelineId), TaskInstanceId: $(system.TaskInstanceId), AuthToken: $(system.AccessToken)}'
    #body: # Required when method != GET && Method != HEAD
    #urlSuffix: # Optional
    #waitForCompletion: 'false' # Options: true, false
    #successCriteria: # Optional
```

Arguments

PARAMETER	COMMENTS
Generic endpoint	Required. Select a Generic service connection. Provides the baseUrl for the call and the authorization to use.
Method	Required. The HTTP method with which the API will be invoked; for example, **GET**, **PUT**, or **UPDATE**.
Headers	Optional. The header in JSON format to be attached to the request sent to the API.
Body	Optional. The request body for the function call in JSON format.

PARAMETER	COMMENTS
URL suffix and parameters	The string to append to the baseUrl from the Generic service connection while making the HTTP call
Completion event	Required. How the task reports completion. Can be **API response** (the default) - completion is when function returns success and success criteria evaluates to true, or **Callback** - the Azure function makes a callback to update the timeline record.
Success criteria	Optional. How to parse the response body for success.
Control options	See Control options

Succeeds if the API returns success and the response body parsing is successful, or when the API updates the timeline record with success.

The **Invoke REST API task** does not perform deployment actions directly. Instead, it allows you to invoke any generic HTTP REST API as part of the automated pipeline and, optionally, wait for it to be completed.

For more information about using this task, see Approvals and gates overview.

Also see this task on GitHub.

Utility: Manual intervention

7/9/2018 • 2 minutes to read • Edit Online

Pause an active deployment within an environment, typically to perform some manual steps or actions, and then continue the automated deployment tasks.

Demands

Can be used in only an agentless phase of a release pipeline.

YAML snippet

```
# Manual Intervention
# Pause deployment and wait for intervention
- task: ManualIntervention@8
  inputs:
    #instructions: # Optional
    #emailRecipients: # Optional
    #onTimeout: 'reject' # Optional. Options: reject, resume
```

Arguments

PARAMETER	COMMENTS
Display name	Required. The name to display for this task.
Instructions	Optional. The instruction text to display to the user when the task is activated.
Notify users	Optional. The list of users that will be notified that the task has been activated.
On timeout	Required. The action to take (reject or resume) if the task times out with no manual intervention. The default is to reject the deployment.
Control options	See Control options

The **Manual Intervention** task does not perform deployment actions directly. Instead, it allows you to pause an active deployment within an environment, typically to perform some manual steps or actions, and then continue the automated deployment tasks. For example, the user may need to edit the details of the current release before continuing; perhaps by entering the values for custom variables used by the tasks in the release.

The **Manual Intervention** task configuration includes an **Instructions** parameter that can be used to provide related information, or to specify the manual steps the user should execute during the Agentless 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.

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.

Utility: PowerShell

7/9/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Run a PowerShell script

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

DotNetFramework

YAML snippet

```
# PowerShell
# Run a PowerShell script on Windows, macOS, or Linux.
- task: PowerShell@2
  inputs:
    #targetType: 'filePath' # Optional. Options: filePath, inline
    #filePath: # Required when targetType == FilePath
    #arguments: # Optional
    #script: '# Write your powershell commands here.Write-Host Hello World# Use the environment variables
input below to pass secret variables to this script.' # Required when targetType == Inline
    #errorActionPreference: 'stop' # Optional. Options: stop, continue, silentlyContinue
    #failOnStderr: false # Optional
    #ignoreLASTEXITCODE: false # Optional
    #workingDirectory: # Optional
```

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 $(Build.SourcesDirectory).
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 pipeline, add this task:

TASK	ARGUMENTS
Utility: PowerShell	Run test.ps1. **Script filename**: `test.ps1`

Write a warning

Set warning message

- Arguments

```
"You've been warned by"
```

- Script

```
Write-Host "$("##vso[task.setvariable variable=WarningMessage]") $($args[0])"
```

Write warning using task.LogIssue

- Script

```
# Writes a warning to build summary and to log in yellow text
Write-Host  "$("##vso[task.logissue type=warning;]") $($env:WarningMessage) $("the task.LogIssue Team
Build logging command.")"
```

Write warning using PowerShell command

- Script

```
# Writes a warning to log preceded by "WARNING: "
Write-Warning "$($env:WarningMessage) $("the Write-Warning PowerShell command.")"
```

Write an error

Set error message

- Arguments

```
"something went wrong."
```

- Script

```
Write-Host "$("##vso[task.setvariable variable=ErrorMessage]") $($args[0])"
```

Write error using task.LogIssue

- Script

```
# Writes an error to the build summary and to the log in red text
Write-Host  "$("##vso[task.logissue type=error;]") $("the task.LogIssue 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

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn about PowerShell scripts?

Scripting with Windows PowerShell

Microsoft Script Center (the Scripting Guys)

Windows PowerShell Tutorial

PowerShell.org

How do I set a variable so that it can be read by subsequent scripts and tasks?

Define and modify your build variables in a script

Define and modify your release variables in a script

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes 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

YAML snippet

```
# Publish Build Artifacts
# Publish build artifacts to Visual Studio Team Services/TFS or a file share
- task: PublishBuildArtifacts@1
  inputs:
    #pathtoPublish: '$(Build.ArtifactStagingDirectory)'
    #artifactName: 'drop'
    #publishLocation: 'Container' # Options: container, filePath
    #targetPath: # Required when publishLocation == FilePath
    #parallel: false # Optional
    #parallelCount: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path to publish	Path to the folder or file you want to publish. The path must be a fully-qualified path or a valid path relative to the root directory of your repository. Typically, you'll specify `$(Build.ArtifactStagingDirectory)`. See Artifacts in Team Build.
Artifact name	Specify the name of the artifact that you want to create. It can be whatever you want. For example: `drop`
Artifact publish location	In most cases, VSTS/TFS (`Server` on **TFS 2018 RTM** and older) is the best and simplest option. Otherwise, choose a file share and then specify a few more arguments (see below). To learn more, see Artifacts in Team Build.
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.

ARGUMENT	DESCRIPTION
Parallel copy (**VSTS**, **TFS 2018**, or newer)	Select whether to copy files in parallel using multiple threads for greater potential throughput. If this setting is not enabled, a single thread will be used.
Parallel count (**VSTS**, **TFS 2018**, or newer)	Enter the degree of parallelism (the number of threads) used to perform the copy. The value must be at least 1 and not greater than 128. Choose a value based on CPU capabilities of the build agent. Typically, 8 is a good starting value.
Control options	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Utility: Publish To Azure Service Bus

7/9/2018 • 2 minutes to read • Edit Online

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 pipeline.

YAML snippet

```
# Publish To Azure Service Bus
# Sends a message to azure service bus using a service connection (no agent required).
- task: PublishToAzureServiceBus@1
  inputs:
    azureSubscription:
    #messageBody: # Optional
    #signPayload: false
    #certificateString: # Required when signPayload == True
    #signatureKey: 'signature' # Optional
    #waitForCompletion: false
```

Arguments

PARAMETER	COMMENTS
Display name	Required. The name to display for this task.
Azure Service Bus Connection	Required. An existing service connection to an Azure Service Bus.
Message body	Required. The text of the message body to send to the Service Bus.
Wait for Task Completion	Optional. Set this option to force the task to halt until a response is received.
Control options	See Control options

Also see this task on GitHub.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Utility: Python Script

7/9/2018 • 2 minutes to read • Edit Online

Run a Python script.

YAML snippet

```
# Python Script
# Run a Python script.
- task: PythonScript@0
  inputs:
    #scriptSource: 'filePath' # Options: filePath, inline
    #scriptPath: # Required when scriptSource == FilePath
    #script: # Required when scriptSource == Inline
    #arguments: # Optional
    #pythonInterpreter: # Optional
    #workingDirectory: # Optional
    #failOnStderr: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Type	(Required) Target script type: File path or Inline
Script Path	(Required when targetType == filePath) Path of the script to execute. Must be a fully qualified path or relative to $(System.DefaultWorkingDirectory).
Script	(Required when targetType == inline) The Python script to run
Arguments	(Optional) Arguments passed to the script execution, available through `sys.argv`.
Python interpreter	(Optional) Absolute path to the Python interpreter to use. If not specified, the task will use the interpreter in PATH.
Working directory	(Optional) undefined
Fail on standard error	(Optional) If this is true, this task will fail if any text are written to the stderr stream.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Utility: Query Azure Monitor Alerts

7/9/2018 • 2 minutes to read • Edit Online

Observe the configured Azure monitor rules for active alerts.

Can be used in only an agentless phase of a release pipeline.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Query Azure Monitor Alerts
# Observe the configured Azure monitor rules for active alerts.
- task: AzureMonitor@0
  inputs:
    connectedServiceNameARM:
    resourceGroupName:
    #resourceType: 'Microsoft.Insights/components' # Options: microsoft.Insights/Components,
microsoft.Web/Sites, microsoft.Storage/StorageAccounts, microsoft.Compute/VirtualMachines
    resourceName:
    alertRules:
```

Arguments

PARAMETER	COMMENTS
Azure subscription	Required. Select an Azure Resource Manager service connection.
Resource group	Required. The resource group being monitored in the subscription.
Resource type	Required. Select the resource type in the selected group.
Resource name	Required. Select the resources of the chosen types in the selected group.
Alert rules	Required. Select from the currently configured alert rules to query for status.
Control options	See Control options

Succeeds if none of the alert rules are activated at the time of sampling.

For more information about using this task, see Approvals and gates overview.

Also see this task on GitHub.

Utility: Query Work Items

7/9/2018 • 2 minutes to read • Edit Online

Ensure the number of matching items returned by a work item query in within the configured thresholds.

Can be used in only an agentless phase of a release pipeline.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# Query Work Items
# Executes a work item query and checks for the number of items returned.
- task: queryWorkItems@0
  inputs:
    queryId:
    #maxThreshold: '0'
    #minThreshold: '0'
```

Arguments

PARAMETER	COMMENTS
Query	Required. Select a work item query within the current project. Can be a built-in or custom query.
Upper threshold	Required. Maximum number of matching workitems for the query. Default value = 0
Lower threshold	Required. Minimum number of matching workitems for the query. Default value = 0
Control options	See Control options

Succeeds if *minimum-threshold* **<=** *#-matching-workitems* **<=** *maximum-threshold*

For more information about using this task, see Approvals and gates overview.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Utility: Service Fabric PowerShell

7/9/2018 • 2 minutes 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.

YAML snippet

```
# Service Fabric PowerShell
# Run a PowerShell script within the context of an Azure Service Fabric cluster connection.
- task: ServiceFabricPowerShell@1
  inputs:
    clusterConnection:
    #scriptType: 'FilePath' # Options: filePath, inlineScript
    #scriptPath: # Optional
    #inline: '# You can write your PowerShell scripts inline here. # You can also pass predefined and custom variables
to this script using arguments' # Optional
    #scriptArguments: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Cluster Connection	The Azure Service Fabric service connection to use to connect and authenticate to the cluster.
Script Type	Specify whether the script is provided as a file or inline in the task.
Script Path	Path to the PowerShell script to run. Can include wildcards and variables. Example: `$(system.defaultworkingdirectory)/**/drop/projectartifacts/**/doc compose.yml` . **Note**: combining compose files is not supported as part of this task.
Script Arguments	Additional parameters to pass to the PowerShell script. Can be either ordinal or named parameters.
Inline Script	The PowerShell commands to run on the build agent. More information
Control options	See Control options

Also see: Service Fabric Compose Deploy task

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Run a shell script using bash

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

sh

YAML snippet

```
- task: ShellScript@2
  inputs:
    scriptPath:
    #args: '' # Optional
    #disableAutoCwd: false # Optional
    #cwd: '' # Optional
    #failOnStandardError: false
```

Arguments

ARGUMENT	DESCRIPTION
Script Path	Relative path from the repo root to the shell script file that you want to run.
Arguments	Arguments that you want to pass to the script.
ADVANCED	
Working Directory	Working directory in which you want to run the script. If you leave it empty it is folder where the script is located.
Fail on Standard Error	Select if you want this task to fail if any errors are written to the StandardError stream.
CONTROL OPTIONS	

Example

Create `test.sh` at the root of your repo:

```
#!/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 pipeline, add this task:

Utility: Shell Script	Run test.bat. • Script Path: `test.sh`

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn about Bash scripts?

Beginners/BashScripting to get started.

Awesome Bash to go deeper.

How do I set a variable so that it can be read by subsequent scripts and tasks?

Define and modify your build variables in a script

Define and modify your release variables in a script

Q: I'm having problems. How can I troubleshoot them?

A: Try this:

1. On the variables tab, add `system.debug` and set it to `true`. Select to allow at queue time.
2. In the explorer tab, view your completed build and click the build step to view its output.

The control options arguments described above can also be useful when you're trying to isolate a problem.

Q: How do variables work? What variables are available for me to use in the arguments?

A: `$(Build.SourcesDirectory)` and `$(Agent.BuildDirectory)` are just a few of the variables you can use. See Variables.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes 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

YAML snippet

```
# Update Service Fabric Manifests
# Automatically updates portions of the application and service manifests within a packaged Service Fabric
application.
- task: ServiceFabricUpdateManifests@2
  inputs:
    #updateType: 'Manifest versions' # Options: manifest Versions, docker Image Settings
    applicationPackagePath:
    #versionSuffix: '.$(Build.BuildNumber)' # Required when updateType == Manifest Versions
    #versionBehavior: 'Append' # Optional. Options: append, replace
    #updateOnlyChanged: false # Required when updateType == Manifest Versions
    #pkgArtifactName: # Optional
    #logAllChanges: true # Optional
    #compareType: 'LastSuccessful' # Optional. Options: lastSuccessful, specific
    #buildNumber: # Optional
    #overwriteExistingPkgArtifact: true # Optional
    #imageDigestsPath: # Required when updateType == Docker Image Settings
```

Arguments

ARGUMENT	DESCRIPTION
Application Package	The location of the Service Fabric application package to be deployed to the cluster. • Example: `$(system.defaultworkingdirectory)/**/drop/applicationpackage` • Can include wildcards and variables.

ARGUMENT	DESCRIPTION

Version Value	The value appended to the versions in the manifest files. Default is `.$(Build.BuildNumber)`.
	
	Tip: You can modify the build number format directly or use a logging command to dynamically set a variable in any format. For example, you can use `$(VersionSuffix)` defined in a PowerShell task:
	
	`$versionSuffix = ".$([DateTimeOffset]::UtcNow.ToString('yyyyMMdd.HHmmss'))"`
	
	`Write-Host "##vso[task.setvariable variable=VersionSuffix;]$versionSuffix"`

Version Behavior	Specify whether to append the version value to existing values in the manifest files, or replace them.
Update only if changed	Select this check box if you want to append the new version suffix to only the packages that have changed from a previous build. If no changes are found, the version suffix from the previous build will be appended. **Note:** By default, the compiler will create different outputs even if you made no changes. Use the deterministic compiler flag to ensure builds with the same inputs produce the same outputs.
Package Artifact Name	The name of the artifact containing the application package from the previous build.
Log all changes	Select this check box to compare all files in every package and log if the file was added, removed, or if its content changed. Otherwise, compare files in a package only until the first change is found for potentially faster performance.
Compare against	Specify whether to compare against the last completed, successful build or against a specific build.
Build Number	If comparing against a specific build, the build number to use.
CONTROL OPTIONS	

Also see: Service Fabric Application Deployment task

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Test: App Center Test

7/9/2018 • 4 minutes to read • Edit Online

Test app packages with Visual Studio App Center.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

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

Arguments

ARGUMENT	DESCRIPTION
Binary application file path	(Required) Relative path from the repo root to the APK or IPA file you want to test.
Artifacts directory	(Required) Where to place the artifacts produced by the prepare step and used by the run step. This directory will be created if it does not exist.
Prepare tests	(Optional) undefined
Test framework	(Required) undefined
Build directory	(Required) Path to directory with Appium tests.
Build directory	(Optional) Path to Espresso output directory.
Test APK path	(Optional) Path to APK file with Espresso tests. If not set, build-dir is used to discover it. Wildcard is allowed.
Project directory	(Required) Path to Calabash workspace directory.
Cucumber config file	(Optional) Path to Cucumber configuration file, usually cucumber.yml.
Profile to run	(Optional) Profile to run. This value must exists in the Cucumber configuration file.
Skip Configuration Check	(Optional) Force running without Cucumber profile.
Build directory	(Required) Path to directory with built test assemblies.
Store file	(Optional) undefined
Store password	(Optional) undefined
Key alias	(Optional) undefined
Key password	(Optional) undefined
Test tools directory	(Optional) Path to directory with Xamarin UI test tools that contains test-cloud.exe.
Signing information	(Optional) Use Signing Infor for signing the test server.
Build directory	(Optional) Path to the build output directory (usually $(ProjectDir)/Build/Products/Debug-iphoneos).
Test IPA path	(Optional) Path to the *.ipa file with the XCUITest tests.
Additional options	(Optional) Additional arguments passed to the App Center test prepare step.

ARGUMENT	DESCRIPTION
Run tests	(Optional) undefined
Authentication method	(Required) Use App Center service connection or enter credentials to connect to Visual Studio App Center.
App Center connection	(Required) Select the service connection for your Visual Studio App Center connection. To create one, click the Manage link and create a new service connection.
App Center username	(Required) Visit https://appcenter.ms/settings/profile to get your username.
App Center password	(Required) Visit https://appcenter.ms/settings/profile to set your password. It can accept variable defined in Build/Release pipelines as '$(passwordVariable)'. You may mark variable type as 'secret' to secure it.
App slug	(Required) The app slug is in the format of {username}/{app_identifier}. To locate {username} and {app_identifier} for an app, click on its name from https://appcenter.ms/apps, and the resulting url is in the format of https://appcenter.ms/users/{username}/apps/{app_identifier}.
Devices	(Required) String to identify what devices this test will run against. Copy and paste this string when you define a new test run from App Center Test beacon.
Test series	(Optional) The series name for organizing test runs (e.g. master, production, beta).
dSYM directory	(Optional) Path to iOS symbol files.
System language	(Required) If your language isn't displayed, select 'Other' and enter its locale below, such as en_US.
Other locale	(Optional) Enter any two-letter ISO-639 language code along with any two-letter ISO 3166 country code in the format [language]_[country], such as en_US.
Additional options for login	(Optional) Additional arguments passed to the App Center login step.
Additional options for run	(Optional) Additional arguments passed to the App Center test run.
Do not wait for test result	(Optional) Execute command asynchronously, exit when tests are uploaded, without waiting for test results.
App Center CLI location	(Optional) Path to the App Center CLI on the build or release agent.
Enable debug output	(Optional) Add --debug to the App Center CLI.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Test: Cloud-based Load Test

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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 pipeline 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

YAML snippet

```
# Cloud-based Load Test
# Runs the load test in the cloud with Visual Studio Team Services
- task: CloudLoadTest@1
  inputs:
    #connectedServiceName: # Optional
    #testDrop: '$(System.DefaultWorkingDirectory)'
    loadTest:
    #activeRunSettings: 'useFile' # Optional. Options: useFile, changeActive
    #runSettingName: # Required when activeRunSettings == ChangeActive
    #testContextParameters: # Optional
    #testSettings: # Optional
    #thresholdLimit: # Optional
    #machineType: '0' # Options: 0, 2
    #resourceGroupName: 'default' # Optional
    #numOfSelfProvisionedAgents: # Optional
```

Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
VSTS connection	The name of a Generic service connection that references the VSTS organization 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 organization 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 Generic service connection.
Test settings file	Required. The path relative to the repository root of the test settings file that specifies the files and data required for the load test such as the test settings, any deployment items, and setup/clean-up scripts. The task will search this path and any subfolders.
Load test files folder	Required. The path of the load test project. The task looks here for the files required for the load test, such as the load test file, any deployment items, and setup/clean-up scripts. The task will search this path and any subfolders.
Load test file	Required. The name of the load test file (such as **myfile.loadtest**) to be executed as part of this task. This allows you to have more than one load test file and choose the one to execute based on the deployment environment or other factors.
Number of permissible threshold violations	Optional. The number of critical violations that must occur for the load test to be deemed unsuccessful, aborted, and marked as failed.
Control options	See Control options

Examples

- Load test your app in the cloud
- Scheduling Load Test Execution

More Information

- Cloud-based Load Testing
- Source code for this task
- Build your Visual Studio solution
- Cloud-based Load Testing Knowledge Base

Related tasks

- Cloud-based Web Performance Test

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

How do I use a Test Settings file?

The **Test settings file** references any setup and cleanup scripts required to execute the load test. For more details see: Using Setup and Cleanup Script in Cloud Load Test

When should I specify the number of permissible threshold violations?

Use the **Number of permissible threshold violations** setting if your load test is not already configured with information about how many violations will cause a failure to be reported. For more details, see: How to: Analyze Threshold Violations Using the Counters Panel in Load Test Analyzer.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Runs the Quick Web Performance Test with VSTS.

The task can be used in a build or release pipeline 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

YAML snippet

```
# Cloud-based Web Performance Test
# Runs a quick web performance test in the cloud with Visual Studio Team Services
- task: QuickPerfTest@1
  inputs:
    #connectedServiceName: # Optional
    websiteUrl:
    testName:
    #vuLoad: '25' # Options: 25, 50, 100, 250
    #runDuration: '60' # Options: 60, 120, 180, 240, 300
    #geoLocation: 'Default' # Optional. Options: default, australia East, australia Southeast, brazil South,
central India, central US, east Asia, east US 2, east US, japan East, japan West, north Central US, north
Europe, south Central US, south India, southeast Asia, west Europe, west US
    #machineType: '0' # Options: 0, 2
    #resourceGroupName: 'default' # Optional
    #numOfSelfProvisionedAgents: # Optional
    #avgResponseTimeThreshold: '0' # Optional
```

Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
VSTS connection	The name of a Generic service connection that references the VSTS organization 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 organization 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 Generic service connection.
Website Url	Required. The URL of the app to test.
Test Name	Required. A name for this load test, used to identify it for reporting and for comparison with other test runs.
User Load	Required. The number of concurrent users to simulate in this test. Select a value from the drop-down list.
Run Duration (sec)	Required. The duration of this test in seconds. Select a value from the drop-down list.
Load Location	The location from which the load will be generated. Select a global Azure location, or **Default** to generate the load from the location associated with your VSTS organization.
Run load test using	Select **Automatically provisioned agents** if you want the cloud-based load testing service to automatically provision agents for running the load tests. The application URL must be accessible from the Internet. Select **Self-provisioned agents** if you want to test apps behind the firewall. You must provision agents and register them against your VSTS organization when using this option. See Testing private/intranet applications using Cloud-based load testing.
Fail test if Avg. Response Time (ms) exceeds	Specify a threshold for the average response time in milliseconds. If the observed response time during the load test exceeds this threshold, the task will fail.
Control options	See Control options

More Information

- Cloud-based Load Testing
- Performance testing video and Q&A

Related tasks

- Cloud-based Load Test
- Cloud-based Apache JMeter Load Test

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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 Code Coverage Results

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Publishes code coverage results that were produced by a build in Cobertura or JaCoCo format.

YAML snippet

```
# Publish Code Coverage Results
# Publish Cobertura or JaCoCo code coverage results from a build
- task: PublishCodeCoverageResults@1
  inputs:
    #codeCoverageTool: 'JaCoCo' # Options: cobertura, jaCoCo
    summaryFileLocation:
    #reportDirectory: # Optional
    #additionalCodeCoverageFiles: # Optional
    #failIfCoverageEmpty: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Code coverage tool	(Required) The tool with which code coverage results are generated.
Summary file	(Required) Path of the summary file containing code coverage statistics, such as line, method, and class coverage. The value may contain minimatch patterns. For example: `$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura/coverage`
Report directory	(Optional) Path of the code coverage HTML report directory. The report directory is published for later viewing as an artifact of the build. The value may contain minimatch patterns. For example: `$(System.DefaultWorkingDirectory)/MyApp/**/site/cobertura`
Additional files	(Optional) File path pattern specifying any additional code coverage files to be published as artifacts of the build. The value may contain minimatch patterns. For example: `$(System.DefaultWorkingDirectory)/**/*.exec`
Fail when code coverage results are missing	(Optional) Available only on VSTS and TFS 2018 and later. Fail the task if code coverage did not produce any results to publish.
CONTROL OPTIONS	

More Information

- Continuous testing scenarios and capabilities

Related tasks

- Publish Test Results

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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

YAML snippet

```
# Publish Test Results
# Publish Test Results to VSTS/TFS
- task: PublishTestResults@2
  inputs:
    #testRunner: 'JUnit' # Options: jUnit, nUnit, vSTest, xUnit
    #testResultsFiles: '**\TEST-*.xml'
    #searchFolder: '$(System.DefaultWorkingDirectory)' # Optional
    #mergeTestResults: false # Optional
    #testRunTitle: # Optional
    #platform: # Optional
    #configuration: # Optional
    #publishRunAttachments: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Test result format	Specify the format of the results files you want to publish. The following formats are supported. - Version 1 of the task: JUnit, xUnit 2, NUnit 2, Visual Studio Test Results (TRX) - Version 2 of the task: JUnit, xUnit 2, NUnit 2, NUnit 3, Visual Studio Test Results (TRX)

ARGUMENT	DESCRIPTION
Test results files	Use this to specify one or more test results files. - Version 1 of the task: You can use a single-folder wildcard (`*`) and recursive wildcards (`**`). For example, `**/TEST-*.xml` searches for all the XML files whose names start with `TEST-` in all subdirectories. Multiple paths can be specified, separated by a semicolon. - Version 2 of the task: Uses the minimatch patterns. For example, `**/TEST-*.xml` searches for all the XML files whose names start with `TEST-` in all subdirectories.
Search folder	Available only in version 2 of the task. Folder to search for the test result files. Default is `$(System.DefaultWorkingDirectory)`
Merge test results	When this option is selected, test results from all the files will be reported against a single test run. If this option is not selected, a separate test run will be created for each test result file.
Test run title	Use this option to provide a name for the test run against which the results will be reported. Variable names declared in the build or release process can be used.
Advanced - Platform	Build platform against which the test run should be reported. For example, `x64` or `x86`. If you have defined a variable for the platform in your build task, use that here.
Advanced - Configuration	Build configuration against which the Test Run should be reported. For example, Debug or Release. If you have defined a variable for configuration in your build task, use that here.
Advanced - Upload test results files	When selected, the task will upload all the test result files as attachments to the test run.
Control options	See Control options

More Information

- Continuous testing scenarios and capabilities

Related tasks

- Visual Studio Test
- Publish Code Coverage Results

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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: Run Functional Tests

7/19/2018 • 8 minutes to read • Edit Online

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This task is deprecated in VSTS and TFS 2018 and later. Use version 2.x or higher of the Visual Studio Test task together with phases to run unit and functional tests on the universal agent.

For more details, see Testing with unified agents and phases.

TFS 2017 and earlier

Run Coded UI tests, Selenium tests, and functional tests on a set of machines using the test agent. Use this task when you want to run tests on remote machines, and you cannot run tests on the build machine.

Demands and prerequisites

This task must be preceded by a **Visual Studio Test Agent Deployment** task.

YAML snippet

```
# Run Functional Tests
# Deprecated: This task and it’s companion task (Visual Studio Test Agent Deployment) are deprecated. Use the 'Visual
Studio Test' task instead. The VSTest task can run unit as well as functional tests. Run tests on one or more agents
using the multi-agent phase setting. Use the 'Visual Studio Test Platform' task to run tests without needing Visual
Studio on the agent. VSTest task also brings new capabilities such as automatically rerunning failed tests.
- task: RunVisualStudioTestsusingTestAgent@1
  inputs:
    testMachineGroup:
    dropLocation:
    #testSelection: 'testAssembly' # Options: testAssembly, testPlan
    #testPlan: # Required when testSelection == TestPlan
    #testSuite: # Required when testSelection == TestPlan
    #testConfiguration: # Required when testSelection == TestPlan
    #sourcefilters: '**\*test*.dll' # Required when testSelection == TestAssembly
    #testFilterCriteria: # Optional
    #runSettingsFile: # Optional
    #overrideRunParams: # Optional
    #codeCoverageEnabled: false # Optional
    #customSlicingEnabled: false # Optional
    #testRunTitle: # Optional
    #platform: # Optional
    #configuration: # Optional
    #testConfigurations: # Optional
    #autMachineGroup: # Optional
```

Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Machines	A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. The maximum is 32 machines (or 32 agents). Can be: - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: `dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.34:5` - An output variable from a previous task.
Test Drop Location	Required. The location on the test machine(s) where the test binaries have been copied by a Windows Machine File Copy or Azure File Copy task. System environment variables from the test agent machines can be used to specify the drop location. Examples: `c:\tests` and `%systemdrive%\Tests`
Test Selection	Required. Whether the tests are to be selected from test assemblies or from a test plan.
Test Assembly	Required when **Test Selection** is set to **Test Assembly**. The test assemblies from which the tests should be executed. Paths are relative to the sources directory. - Separate multiple paths with a semicolon. - Default is `**\*test*.dll` - For JavaScript tests, enter the path and name of the **.js** files containing the tests. - Wildcards can be used. Example: `**\commontests\*test*.dll; **\frontendtests\*test*.dll`
Test Filter criteria	Optional when **Test Selection** is set to **Test Assembly**. A filter to specify the tests to execute within the test assembly files. Works the same way as the `/TestCaseFilter` option of vstest.console.exe. Example: **Priority=1 | Name=MyTestMethod**
Test Plan	Required if **Test Suite** is not specified when **Test Selection** is set to **Test Plan**. Select a test plan already configured for this organization.
Test Suite	Required if **Test Plan** is not specified when **Test Selection** is set to **Test Plan**. Select a test suite from the selected test plan.
Test Configuration	Optional when **Test Selection** is set to **Test Plan**. Select a test configuration from the selected test plan.
Run Settings File	Optional. The path to a `.runsettings` or `.testsettings` file on the build machine. Can be the path to a file in the repository or a file on disk. Use `$(Build.SourcesDirectory)` to specify the project root folder.
Override Test Run Parameters	Optional. A string containing parameter overrides for parameters defined in the `TestRunParameters` section of the `.runsettings` file. Example: `Platform=$(platform);Port=8080`
Code Coverage Enabled	When set, the task will collect code coverage information during the run and upload the results to the server. Supported for .NET and C++ projects only.
Distribute tests by number of machines	When checked, distributes tests based on the number of machines, instead of distributing tests at the assembly level, irrespective of the container assemblies passed to the task.

ARGUMENT	DESCRIPTION
Test Run Title	Optional. A name for this test run, used to identify it for reporting and in comparison with other test runs.
Platform	Optional. The build platform against which the test run should be reported. Used only for reporting. - If you are using the **Build - Visual Studio** template, this is automatically defined, such as `x64` or `x86` - If you have defined a variable for platform in your build task, use that here.
Configuration	Optional. The build configuration against which the test run should be reported. Used only for reporting. - If you are using the **Build - Visual Studio** template, this is automatically defined, such as `Debug` or `Release` - If you have defined a variable for configuration in your build task, use that here.
Test Configurations	Optional. A string that contains the filter(s) to report the configuration on which the test case was run. Used only for reporting with Microsoft Test Manager (MTM). - Syntax: {expression for test method name(s)} **:** {configuration ID from MTM} - Example: `FullyQualifiedName~Chrome:12` to report all test methods that have **Chrome** in the **Fully Qualified Name** and map them to configuration ID **12** defined in MTM. - Use `DefaultTestConfiguration:{Id}` as a catch-all.
Application Under Test Machines	A list of the machines on which the Application Under Test (AUT) is deployed, or on which a specific process such as W3WP.exe is running. Used to collect code coverage data from these machines. Use this in conjunction with the **Code Coverage Enabled** setting. The list can be a comma-delimited list of machine names or an output variable from an earlier task.
Control options	See Control options

> The task supports a maximum of 32 machines/agents.

Scenarios

Typical scenarios include:

- Tests that require additional installations on the test machines, such as different browsers for Selenium tests
- Coded UI tests
- Tests that require a specific operating system configuration
- To execute a large number of unit tests more quickly by using multiple test machines

Use this task to:

- Run automated tests against on-premises standard environments
- Run automated tests against existing Azure environments
- Run automated tests against newly provisioned azure environments

You can run unit tests, integration tests, functional tests - in fact any test that you can execute using the Visual Studio test runner (vstest).

Using multiple machines in a Machine Group enables the task to run parallel distributed execution of tests. Parallelism is at the test assembly level, not at individual test level.

These scenarios are supported for:

- **TFS on-premises and VSTS**
- **Build agents**
  - Hosted and on-premises agents.
  - The build agent must be able to communicate with all test machines. If the test machines are on-premises behind a firewall, the hosted build agents cannot be used.
  - The build agent must have access to the Internet to download test agents. If this is not the case, the test agent must be manually downloaded and deployed to a network location that is accessible by the build agent, and a **Visual Studio Test Agent Deployment** task used with an appropriate path for the **Test Agent Location** parameter. Automatic checking for new test agent versions is not supported in this topology.
- **CI/CD workflow**
  - The Build-Deploy-Test (BDT) tasks are supported in both build and release pipelines.
- **Machine group configuration**
  - Only Windows machines are supported when using BDT tasks inside a Machine Group. Using Linux, macOS, or other platforms inside a Machine Group with BDT tasks is not supported.
  - Installing any version or release of Visual Studio on any of the test machines is not supported.
  - Installing an older version of the test agent on any of the test machines is not supported.
- **Test machine topologies**
  - Azure-based test machines are fully supported, both existing test machines and newly provisioned machines.
  - Domain-joined test machines are supported.
  - Workgroup-joined test machines must have HTTPS authentication enabled and configured during creation of the Machine Group.
  - Test agent machines must have network access to the Team Foundation Server instance. Test machines isolated on the network are not supported.
- **Usage Error Conditions**
  - Running tests across different Machine Groups, and running builds (with any BDT tasks) in parallel against these Machine Groups is not supported.
  - Cancelling an in-progress build or release with BDT tasks is not supported. If you do so, subsequent builds may not behave as expected.
  - Cancelling an in-progress test run queued through BDT tasks is not supported.
  - Configuring a test agent and running tests under a non-administrative account or under a service account is not supported.

More information

- Using the Visual Studio Agent Deployment task on machines not connected to the internet
- Set up environments to run continuous test tasks with your builds
- Testing in Continuous Integration and Continuous Deployment Workflows

Related tasks

- Deploy Azure Resource Group
- Azure File Copy
- Windows Machine File Copy
- PowerShell on Target Machines
- Visual Studio Test Agent Deployment

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

How do I create an Azure Resource Group for testing?

See Using the Azure Portal to manage your Azure resources and Azure Resource Manager - Creating a Resource Group and a VNET.

Where can I get more information about the Run Settings file?

See Configure unit tests by using a .runsettings file

Where can I get more information about overriding settings in the Run Settings file?

See Supplying Run Time Parameters to Tests

How can I customize code coverage analysis and manage inclusions and exclusions

See Customize Code Coverage Analysis

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

::: moniker-end

Help and support

- Report problems through the Developer Community
- Send suggestions on UserVoice

Test: Run Distributed Tests

7/9/2018 • 3 minutes to read • Edit Online

Deprecated: This task and its companion task (Visual Studio Test Agent Deployment) are deprecated. Use the 'Visual Studio Test' task instead. The VSTest task can run unit as well as functional tests. Run tests on one or more agents using the multi-agent phase setting. Use the 'Visual Studio Test Platform' task to run tests without needing Visual Studio on the agent. VSTest task also brings new capabilities such as automatically rerunning failed tests.

YAML snippet

```
# Run Functional Tests
# Deprecated: This task and it’s companion task (Visual Studio Test Agent Deployment) are deprecated. Use the
'Visual Studio Test' task instead. The VSTest task can run unit as well as functional tests. Run tests on one
or more agents using the multi-agent phase setting. Use the 'Visual Studio Test Platform' task to run tests
without needing Visual Studio on the agent. VSTest task also brings new capabilities such as automatically
rerunning failed tests.
- task: RunVisualStudioTestsusingTestAgent@1
  inputs:
    testMachineGroup:
    dropLocation:
    #testSelection: 'testAssembly' # Options: testAssembly, testPlan
    #testPlan: # Required when testSelection == TestPlan
    #testSuite: # Required when testSelection == TestPlan
    #testConfiguration: # Required when testSelection == TestPlan
    #sourcefilters: '**\*test*.dll' # Required when testSelection == TestAssembly
    #testFilterCriteria: # Optional
    #runSettingsFile: # Optional
    #overrideRunParams: # Optional
    #codeCoverageEnabled: false # Optional
    #customSlicingEnabled: false # Optional
    #testRunTitle: # Optional
    #platform: # Optional
    #configuration: # Optional
    #testConfigurations: # Optional
    #autMachineGroup: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Machines	(Required) Provide a comma separated list of machine IP addresses or FQDNs. Eg: `dbserver.fabrikam.com or 192.168.12.34` Or provide output variable of other tasks. Eg: `$(variableName)` Or provide a Machine Group Name
Test Drop Location	(Required) Location where the test binaries have been dropped in the agent machine(s) as part of the 'Windows Machine File Copy' or 'Azure File Copy' task. System Environment Variables can also be used in location string. e.g., `%systemdrive%\Tests`, `%temp%\DropLocation` etc.
Test Selection	(Required) Select the way you want to run tests : using Test Assemblies or using Test Plan.

ARGUMENT	DESCRIPTION
Test Plan	(Required) Select a Test Plan.
Test Suite	(Required) Select Test Suites from the Test Plan.
Test Configuration	(Required) Select Test Configuration.
Test Assembly	(Required) Test binaries to run tests on. Wildcards can be used. For example, `**\*test*.dll;` for all dlls containing 'test' in their name.
Test Filter criteria	(Optional) Optionally specify the test case filter criteria. For example, `Owner=james&Priority=1`
Run Settings File	(Optional) Path to runsettings or testsettings file to use with the tests.
Override Test Run Parameters	(Optional) Override parameters defined in the `TestRunParameters` section of runsettings file or `Properties` section of testsettings file. For example: `AppURL=$(DeployURL);Port=8080`. Note: Properties specified in testsettings file can be accessed via the TestContext using Test Agent 2017 Update 4 or higher.
Code Coverage Enabled	(Optional) Whether code coverage needs to be enabled.
Distribute tests by number of machines	(Optional) Tests will be distributed based on the number of machines provided instead of number of test containers. Note that tests within a dll might also be distributed to multiple machines.
Test Run Title	(Optional) Provide a name for the Test Run.
Platform	(Optional) Platform against which the tests should be reported. If you have defined a variable for platform in your build task, use that here.
Configuration	(Optional) Configuration against which the tests should be reported. If you have defined a variable for configuration in your build task, use that here.
Test Configurations	(Optional) Optionally associate a test case filter against a test configuration ID. Syntax: <Filter1>:<Id1>;DefaultTestConfiguration:<Id3>. For example: `FullyQualifiedName~Chrome:12`
Application Under Test Machines	(Optional) Comma separated list of machines or output variable or Machine Group Name on which server processes such as W3WP.exe is running
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Test: Cloud-based Apache JMeter Load Test

7/9/2018 • 2 minutes to read • Edit Online

Runs the Apache JMeter load test in cloud

YAML snippet

```
# Cloud-based Apache JMeter Load Test
# Runs the Apache JMeter load test in cloud
- task: ApacheJMeterLoadTest@1
  inputs:
    #connectedServiceName: # Optional
    testDrop:
    #loadTest: 'jmeter.jmx'
    #agentCount: '1' # Options: 1, 2, 3, 4, 5
    #runDuration: '60' # Options: 60, 120, 180, 240, 300
    #geoLocation: 'Default' # Optional. Options: default, australia East, australia Southeast, brazil South,
central India, central US, east Asia, east US 2, east US, japan East, japan West, north Central US, north
Europe, south Central US, south India, southeast Asia, west Europe, west US
    #machineType: '0' # Optional. Options: 0, 2
```

Arguments

ARGUMENT	DESCRIPTION
VS Team Services Connection	(Optional) Select a previously registered service connection to talk to the cloud-based load test service. Choose 'Manage' to register a new connection.
Apache JMeter test files folder	(Required) Relative path from repo root where the load test files are available.
Apache JMeter file	(Required) The Apache JMeter test filename to be used under the load test folder specified above.
Agent Count	(Required) Number of test agents (dual-core) used in the run.
Run Duration (sec)	(Required) Load test run duration in seconds.
Run load test using	(Optional) undefined
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Test: Visual Studio Test

7/9/2018 • 7 minutes to read • Edit Online

Run unit and functional tests (Selenium, Appium, Coded UI test, etc.) using the Visual Studio Test runner. Test frameworks that have a Visual Studio test adapter such as xUnit, NUnit, Chutzpah, etc. can also be run. Tests can be distributed on multiple agents using this task (version 2).

YAML snippet

```
# Visual Studio Test
# Run unit and functional tests (Selenium, Appium, Coded UI test, etc.) using the Visual Studio Test runner.
Test frameworks that have a Visual Studio test adapter such as xUnit, NUnit, Chutzpah, etc. can also be run.
Tests can be distributed on multiple agents using this task (version 2).
- task: VSTest@2
  inputs:
    #testSelector: 'testAssemblies' # Options: testAssemblies, testPlan, testRun
    #testAssemblyVer2: '**\*test*.dll!**\*TestAdapter.dll!**\obj\**' # Required when testSelector ==
TestAssemblies
    #testPlan: # Required when testSelector == TestPlan
    #testSuite: # Required when testSelector == TestPlan
    #testConfiguration: # Required when testSelector == TestPlan
    #tcmTestRun: '$(test.RunId)' # Optional
    #searchFolder: '$(System.DefaultWorkingDirectory)'
    #testFiltercriteria: # Optional
    #runOnlyImpactedTests: False # Optional
    #runAllTestsAfterXBuilds: '50' # Optional
    #uiTests: false # Optional
    #vstestLocationMethod: 'version' # Optional. Options: version, location
    #vsTestVersion: 'latest' # Optional. Options: latest, 15.0, 14.0, toolsInstaller
    #vstestLocation: # Optional
    #runSettingsFile: # Optional
    #overrideTestrunParameters: # Optional
    #pathtoCustomTestAdapters: # Optional
    #runInParallel: False # Optional
    #runTestsInIsolation: False # Optional
    #codeCoverageEnabled: False # Optional
    #otherConsoleOptions: # Optional
    #distributionBatchType: 'basedOnTestCases' # Optional. Options: basedOnTestCases, basedOnExecutionTime,
basedOnAssembly
    #batchingBasedOnAgentsOption: 'autoBatchSize' # Optional. Options: autoBatchSize, customBatchSize
    #customBatchSizeValue: '10' # Required when distributionBatchType == BasedOnTestCases &&
BatchingBasedOnAgentsOption == CustomBatchSize
    #batchingBasedOnExecutionTimeOption: 'autoBatchSize' # Optional. Options: autoBatchSize,
customTimeBatchSize
    #customRunTimePerBatchValue: '60' # Required when distributionBatchType == BasedOnExecutionTime &&
BatchingBasedOnExecutionTimeOption == CustomTimeBatchSize
    #dontDistribute: False # Optional
    #testRunTitle: # Optional
    #platform: # Optional
    #configuration: # Optional
    #publishRunAttachments: true # Optional
    #rerunFailedTests: False # Optional
    #rerunType: 'basedOnTestFailurePercentage' # Optional. Options: basedOnTestFailurePercentage,
basedOnTestFailureCount
    #rerunFailedThreshold: '30' # Optional
    #rerunFailedTestCasesMaxLimit: '5' # Optional
    #rerunMaxAttempts: '3' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Select tests using	(Required) • **Test assembly:** Use this option to specify one or more test assemblies that contain your tests. You can optionally specify a filter criteria to select only specific tests. • **Test plan:** Use this option to run tests from your test plan that have an automated test method associated with it. • **Test run:** Use this option when you are setting up an environment to run tests from the Test hub. This option should not be used when running tests in a continuous integration / continuous deployment (CI/CD) pipeline.
Test assemblies	(Required) Run tests from the specified files. Ordered tests and webtests can be run by specifying the .orderedtest and .webtest files respectively. To run .webtest, Visual Studio 2017 Update 4 or higher is needed. The file paths are relative to the search folder. Supports multiple lines of minimatch patterns. More Information
Test plan	(Required) Select a test plan containing test suites with automated test cases.
Test suite	(Required) Select one or more test suites containing automated test cases. Test case work items must be associated with an automated test method. [Learn more.](https://go.microsoft.com/fwlink/?linkid=847773
Test configuration	(Required) Select Test Configuration.
Test Run	(Optional) Test run based selection is used when triggering automated test runs from the test hub. This option cannot be used for running tests in the CI/CD pipeline.
Search folder	(Required) Folder to search for the test assemblies.
Test filter criteria	(Optional) Additional criteria to filter tests from Test assemblies. For example: `Priority=1|Name=MyTestMethod`. More information
Run only impacted tests	(Optional) Automatically select, and run only the tests needed to validate the code change. More information
Number of builds after which all tests should be run	(Optional) Number of builds after which to automatically run all tests. Test Impact Analysis stores the mapping between test cases and source code. It is recommended to regenerate the mapping by running all tests, on a regular basis.

ARGUMENT	DESCRIPTION
Test mix contains UI tests	(Optional) To run UI tests, ensure that the agent is set to run in interactive mode. Setting up an agent to run interactively must be done before queueing the build / release. Checking this box does **not** configure the agent in interactive mode automatically. This option in the task is to only serve as a reminder to configure agent appropriately to avoid failures. Hosted Windows agents from the VS 2015 and 2017 pools can be used to run UI tests. More information.
Select test platform using	(Optional) undefined
Test platform version	(Optional) The version of Visual Studio test to use. If latest is specified it chooses Visual Studio 2017 or Visual Studio 2015 depending on what is installed. Visual Studio 2013 is not supported. To run tests without needing Visual Studio on the agent, use the 'Installed by tools installer' option. Be sure to include the 'Visual Studio Test Platform Installer' task to acquire the test platform from nuget.
Path to vstest.console.exe	(Optional) Optionally supply the path to VSTest.
Settings file	(Optional) Path to runsettings or testsettings file to use with the tests.
Override test run parameters	(Optional) Override parameters defined in the `TestRunParameters` section of runsettings file or `Properties` section of testsettings file. For example: `-key1 value1 -key2 value2`. Note: Properties specified in testsettings file can be accessed via the TestContext using Visual Studio 2017 Update 4 or higher
Path to custom test adapters	(Optional) Directory path to custom test adapters. Adapters residing in the same folder as the test assemblies are automatically discovered.
Run tests in parallel on multi-core machines	(Optional) If set, tests will run in parallel leveraging available cores of the machine. This will override the MaxCpuCount if specified in your runsettings file. Click here to learn more about how tests are run in parallel.
Run tests in isolation	(Optional) Runs the tests in an isolated process. This makes vstest.console.exe process less likely to be stopped on an error in the tests, but tests might run slower. This option currently cannot be used when running with the multi-agent phase setting.
Code coverage enabled	(Optional) Collect code coverage information from the test run.

ARGUMENT	DESCRIPTION
Other console options	(Optional) Other console options that can be passed to vstest.console.exe, as documented here. These options are not supported and will be ignored when running tests using the 'Multi agent' parallel setting of an agent phase or when running tests using 'Test plan' option. The options can be specified using a settings file instead.
Batch tests	(Optional) A batch is a group of tests. A batch of tests runs at a time and results are published for that batch. If the phase in which the task runs is set to use multiple agents, each agent picks up any available batches of tests to run in parallel. **Based on number of tests and agents:** Simple batching based on the number of tests and agents participating in the test run. **Based on past running time of tests:** This batching considers past running time to create batches of tests such that each batch has approximately equal running time. **Based on test assemblies:** Tests from an assembly are batched together.
Batch options	(Optional) Simple batching based on the number of tests and agents participating in the test run. When the batch size is automatically determined, each batch contains `(total number of tests / number of agents)` tests. If a batch size is specified, each batch will contain the specified number of tests.
Number of tests per batch	(Required) Specify batch size
Batch options	(Optional) This batching considers past running time to create batches of tests such that each batch has approximately equal running time. Quick running tests will be batched together, while longer running tests may belong to a separate batch. When this option is used with the multi-agent phase setting, total test time is reduced to a minimum.
Running time (sec) per batch	(Required) Specify the running time (sec) per batch
Do not distribute tests and replicate instead when multiple agents are used in the phase	(Optional) Choosing this option will not distribute tests across agents when the task is running in a multi-agent phase. Each of the selected test(s) will be repeated on each agent. The option is not applicable when the agent phase is configured to run with no parallelism or with the multi-config option.
Test run title	(Optional) Provide a name for the test run.
Build platform	(Optional) Build platform against which the tests should be reported. If you have defined a variable for platform in your build task, use that here.

ARGUMENT	DESCRIPTION
Build configuration	(Optional) Build configuration against which the tests should be reported. If you have defined a variable for configuration in your build task, use that here.
Upload test attachments	(Optional) Opt in/out of publishing run level attachments.
Rerun failed tests	(Optional) Selecting this option will rerun any failed tests until they pass or the maximum # of attempts is reached.
Do not rerun if test failures exceed specified threshold	(Optional) Use this option to avoid rerunning tests when failure rate crosses the specified threshold. This is applicable if any environment issues leads to massive failures. You can specify % failures or # of failed tests as a threshold.
% failure	(Optional) Use this option to avoid rerunning tests when failure rate crosses the specified threshold. This is applicable if any environment issues leads to massive failures.
# of failed tests	(Optional) Use this option to avoid rerunning tests when number of failed test cases crosses specified limit. This is applicable if any environment issues leads to massive failures.
Maximum # of attempts	(Optional) Specify the maximum # of times a failed test should be retried. If a test passes before the maximum # of attempts is reached, it will not be rerun further.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Test: Visual Studio Test Agent Deployment

7/9/2018 • 8 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

This task is deprecated in VSTS and TFS 2018 and later. Use version 2.x or higher of the Visual Studio Test task together with phases to run unit and functional tests on the universal agent.

For more details, see Testing with unified agents and phases.

TFS 2017 and earlier

Deploy and configure the test agent to run tests on a set of machines.

The test agent deployed by this task can collect data or run distributed tests using the Visual Studio Test task.

Demands and prerequisites

This task requires the target computer to have:

- Windows 7 Service Pack 1 or Windows 2008 R2 Service Pack 2 or higher
- .NET 4.5 or higher
- PSRemoting enabled by running the Enable-PSRemoting PowerShell script

Windows Remote Management (WinRM)

This task uses Windows Remote Management (WinRM) to access on-premises physical computers or virtual computers that are domain-joined or workgroup-joined.

To set up WinRM for on-premises physical computers or virtual machines

Follow the steps described in domain-joined

To set up WinRM for Microsoft Azure Virtual Machines

Azure Virtual Machines require WinRM to use the HTTPS protocol. You can use a self-signed Test Certificate. In this case, the automation agent will not validate the authenticity of the certificate as being issued by a trusted certification authority.

- **Azure Classic Virtual Machines**. When you create a classic virtual machine from the Azure portal, the virtual machine is already set up for WinRM over HTTPS, with the default port 5986 already opened in the firewall and a self-signed certificate installed on the machine. These virtual machines can be accessed with no further configuration required. Existing Classic virtual machines can be also selected by using the Azure Resource Group Deployment task.
- **Azure Resource Group**. If you have an Azure Resource Group already defined in the Azure portal, you must configure it to use the WinRM HTTPS protocol. You need to open port 5986 in the firewall, and install a self-signed certificate.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the Azure Resource Group Deployment task. This task has a checkbox named **Enable Deployment Pre-requisites**. Select this to automatically set up the WinRM HTTPS protocol on the virtual machines, open port 5986 in the firewall, and install a test certificate. The virtual machines are then ready for use in the deployment task.

YAML snippet

```
# Visual Studio Test Agent Deployment
# Deprecated: This task and it’s companion task (Run Functional Tests) are deprecated. Use the 'Visual Studio Test'
task instead. The VSTest task can run unit as well as functional tests. Run tests on one or more agents using the
multi-agent phase setting. Use the 'Visual Studio Test Platform' task to run tests without needing Visual Studio on
the agent. VSTest task also brings new capabilities such as automatically rerunning failed tests.
- task: DeployVisualStudioTestAgent@2
  inputs:
    testMachines:
    adminUserName:
    adminPassword:
    #winRmProtocol: 'Http' # Options: http, https
    #testCertificate: true # Optional
    machineUserName:
    machinePassword:
    #runAsProcess: false # Optional
    #isDataCollectionOnly: false # Optional
    #testPlatform: '14.0' # Optional. Options: 15.0, 14.0
    #agentLocation: # Optional
    #updateTestAgent: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Machines	A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. The maximum is 32 machines (or 32 agents). Can be: - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: `dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.34:5` - An output variable from a previous task.
Admin Login	The username of either a domain or a local administrative account on the target host(s). This parameter is required when used with a list of machines. It is optional when specifying a machine group and, if specified, overrides the credential settings defined for the machine group. - Formats such as **username**, **domain\username**, **machine-name\username**, and **.\username** are supported. - UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.
Password	The password for the administrative account specified above. This parameter is required when used with a list of machines. It is optional when specifying a machine group and, if specified, overrides the credential settings defined for the machine group. Consider using a secret variable global to the build or release pipeline to hide the password. Example: `$(passwordVariable)`
Protocol	The protocol that will be used to connect to the target host, either **HTTP** or **HTTPS**.
Agent Configuration - Username	Required. The username that the test agent will use. Must be an account on the test machines that has administrative permissions. - 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.
Agent Configuration - Password	Required. The password for the **Username** for the test agent. To protect the password, create a variable and use the "padlock" icon to hide it.

ARGUMENT	DESCRIPTION
Agent Configuration - Run UI tests	When set, the test agent will run as an interactive process. This is required when interacting with UI elements or starting applications during the tests. For example, Coded UI or Selenium tests that are running on full fidelity browsers will require this option to be set.
Agent Configuration - Enable data collection only	When set, the test agent will return previously collected data and not re-run the tests. At present this is only available for Code Coverage. Also see Q&A section below.
Advanced - Test agent version	The version of the test agent to use.
Advanced - Test agent location	Optional. The path to the test agent (vstf_testagent.exe) if different from the default path. - If you use a copy of the test agent located on your local computer or network, specify the path to that instance. - The location must be accessible by either the build agent (using the identity it is running under) or the test agent (using the identity configured above). - For Azure test machines, the web location can be used.
Advanced - Update test agent	If set, and the test agent is already installed on the test machines, the task will check if a new version of the test agent is available.
Control options	See Control options

> The task supports a maximum of 32 machines/agents.

Supported scenarios

Use this task for:

- Running automated tests against on-premises standard environments
- Running automated tests against existing Azure environments
- Running automated tests against newly provisioned Azure environments

The supported options for these scenarios are:

- **TFS**
  - On-premises and VSTS
- **Build and release agents**
  - Hosted and on-premises agents are supported.
  - The agent must be able to communicate with all test machines. If the test machines are on-premises behind a firewall, a VSTS Microsoft-hosted agent cannot be used because it will not be able to communicate with the test machines.
  - The agent must have Internet access to download test agents. If this is not the case, the test agent must be manually downloaded, uploaded to a network location accessible to the agent, and the **Test Agent Location** parameter used to specify the location. The user must manually check for new versions of the agent and update the test machines.
- **Continuous integration/continuous deployment workflows**
  - Build/deploy/test tasks are supported in both Build and Release Management workflows.
- **Machine group configuration**
  - Only Windows-based machines are supported inside a machine group for build/deploy/test tasks. Linux, macOS, or other platforms are not supported inside a machine group.
  - Installing any version of Visual Studio on any of the test machines is not supported.

- Installing any older version of the test agent on any of the test machines is not supported.

- **Test machine topologies**
  - Azure-based test machines are fully supported, both existing test machines and newly provisioned test machines.
  - Machines with the test agent installed must have network access to the TFS instance in use. Network-isolated test machines are not supported.
  - Domain-joined test machines are supported.
  - Workgroup-joined test machines must use HTTPS authentication configured during machine group creation.

- **Usage Error Conditions**
  - Using the same test machines across different machine groups, and running builds (with any build/deploy/test tasks) in parallel against those machine groups is not supported.
  - Cancelling an in-progress build or release that contains any build/deploy/test tasks is not supported. If you do cancel, behavior of subsequent builds may be unpredictable.
  - Cancelling an ongoing test run queued through build/deploy/test tasks is not supported.
  - Configuring the test agent and running tests as a non-administrator, or by using a service account, is not supported.
  - Running tests for Universal Windows Platform apps is not supported. Use the Visual Studio Test task to run these tests.

Example

- Testing in Continuous Integration and Continuous Deployment Workflows

More information

- Using the Visual Studio Agent Deployment task on machines not connected to the internet
- Set up automated testing for your builds
- Source code for this task

Related tasks

- Visual Studio Test
- Azure File Copy
- Windows Machine File Copy
- PowerShell on Target Machines

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

When would I use the Enable Data Collection Only option?

An example would be in a client-server application model, where you deploy the test agent on the servers and use another task to deploy the test agent to test machines. This enables you to collect data from both server and client machines without triggering the execution of tests on the server machines.

How do I create an Azure Resource Group for testing?

See Using the Azure Portal to manage your Azure resources and Azure Resource Manager - Creating a Resource Group and a VNET.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

::: moniker-end

Help and support

- Report problems through the Developer Community
- Send suggestions on UserVoice

Test: Xamarin Test Cloud

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Test mobile apps with Xamarin Test Cloud using Xamarin.UITest

> **NOTE:** This task is deprecated. Use the App Center Test task instead.

Demands

None

YAML snippet

```
# Xamarin Test Cloud
# [Depreciated] Testing mobile apps with Xamarin Test Cloud using Xamarin.UITest - recommended task is now
AppCenterTest
- task: XamarinTestCloud@1
  inputs:
    appFile:
    #dsymFile: # Optional
    teamApiKey:
    email:
    devices:
    #series: 'master'
    testAssemblyDirectory:
    #parallelizationOption: 'none' # Options: none, --Fixture-Chunk, --Test-Chunk
    #localeOption: 'en_US' # Options: da_DK, nl_NL, en_GB, en_US, fr_FR, de_DE, ja_JP, ru_RU, es_MX, es_ES,
user
    #userDefinedLocale: # Optional
    #testCloudFile: '**/packages/**/tools/test-cloud.exe'
    #optionalArgs: # Optional
    #publishNUnitResults: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
App File	- If you want to test a single app, choose **...** and select the .apk file. - If you want to test multiple apps, specify a match pattern filter. You can use a single-folder wildcard (`*`) and recursive wildcards (`**`). For example, `**/*.apk` searches for all .apk files in all subdirectories.
dSYM File	To make crash logs easier to read, upload a .dSYM file that is associated with your app. Specify a path relative to the .ipa file. You can use a match pattern filter. For example: `*.dSYM` **Note:** This argument applies only to iOS apps.

ARGUMENT	DESCRIPTION
Team API Key	Your Xamarin Test Cloud Team API key. To get it, go to your Xamarin Test Cloud account, and then to **Teams & Apps**. **Important:** Use a secret variable to avoid exposing this value.
User Email	Email address of your Xamarin Test Cloud account.
Devices	The devices string generated by Xamarin Test Cloud. To get this string: - Go to your Xamarin Test Cloud account. - Choose **New Test Run**. - Step through the wizard. - When you reach the final page, copy the ID that follows the **--devices** option.
Series	Series name for the test run. For example, `master`, `production`, or `beta`. See Xamarin: Creating A Test Run for a Team.
Test Assembly Directory	Relative path to the folder that contains the test assemblies. For example: `SolutionName/TestsProjectName/bin/Release`.
Advanced - Parallelization	Select None, By test fixture, or By test method.
Advanced - System Language	Select your language. If it isn't displayed, select Other and then enter its locale below. For example: `en_AU`.
Advanced - test-cloud.exe Location	Location of test-cloud.exe. In most cases leave this set to the default value.
Advanced - Optional Arguments	(Optional) Arguments passed to test-cloud.exe. See Submitting UITests at the Command Line.
Advanced - Publish results to VSO/TFS	Select if you want to pass the **--nunit-xml** option to test-cloud.exe so that results from the NUnit xml file are be published to TFS or VSTS.
Control options	See Control options

Example

Build your Xamarin app

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

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

7/9/2018 • 2 minutes 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

YAML snippet

```
# CocoaPods
# CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects. This task runs 'pod install'.
- task: CocoaPods@0
  inputs:
    #workingDirectory: # Optional
    forceRepoUpdate:
    #projectDirectory: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Working Directory	Working directory. If you leave it blank, the working directory is the root of your repo.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

What other kinds of apps can I build?

Build your app

What other kinds of build tasks are available?

Specify your build tasks

How do we protect our codebase from build breaks?

- Git: Improve code quality with branch policies with an option to require that code builds before it can be merged to a branch. This option is not available for GitHub repos.
- TFVC: Use gated check-in.

How do I modify other parts of my build pipeline?

- Specify your build tasks to run tests, scripts, and a wide range of other processes.
- Specify build options such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- Specify the repository to pick the source of the build and modify options such as how the agent workspace is cleaned.
- Set build triggers to modify how your CI builds run and to specify scheduled builds.
- Specify build retention policies to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent jobs. See Concurrent build and release jobs in VSTS.

How do I see what has changed in my build pipeline?

View the change history of your build pipeline

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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: Conda Environment

7/9/2018 • 2 minutes to read • Edit Online

Create and activate a Conda environment.

This task will create a Conda environment and activate it for subsequent build tasks.

If the task finds an existing environment with the same name, the task will simply reactivate it. This is possible on self-hosted agents. To recreate the environment and reinstall any of its packages, set the "Clean the environment" option.

Running with the "Update to the latest Conda" option will attempt to update Conda before creating or activating the environment. If you are running a self-hosted agent and have configured a Conda installation to work with the task, this may result in your Conda installation being updated.

> **NOTE**
> Microsoft-hosted agents won't have Conda in their `PATH` by default. You will need to run this task in order to use Conda.

After running this task, `PATH` will contain the binary directory for the activated environment, followed by the binary directories for the Conda installation itself. You can run scripts as subsequent build tasks that run Python, Conda, or the command-line utilities from other packages you install. For example, you can run tests with pytest or upload a package to Anaconda Cloud with the Anaconda client.

> **TIP**
> After running this task, the environment will be "activated," and packages you install by calling `conda install` will get installed to this environment.

Demands

None

Prerequisites

- A Microsoft-hosted agent, or a self-hosted agent with Anaconda or Miniconda installed.
- If using a self-hosted agent, you must either add the `conda` executable to `PATH` or set the `CONDA` environment variable to the root of the Conda installation.

YAML snippet

```
# Conda Environment
# Create and activate a Conda environment.
- task: CondaEnvironment@1
  inputs:
    #createCustomEnvironment: # Optional
    #environmentName: # Required when createCustomEnvironment == True
    #packageSpecs: 'python=3' # Optional
    #updateConda: # Optional
    #installOptions: # Optional
    #createOptions: # Optional
    #cleanEnvironment: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Environment name	Name of the Conda environment to create and activate.
Package specs	Space-delimited list of packages to install when creating the environment.
Update to the latest Conda	Update Conda to the latest version. This applies to the Conda installation found in `PATH` or at the path specified by the `CONDA` environment variable.

Advanced

ARGUMENT	DESCRIPTION
Environment creation options	Space-delimited list of other options to pass to the `conda create` command.
Clean the environment	Delete the environment and recreate it if it already exists. If not selected, the task will reactivate an existing environment.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 can I configure a self-hosted agent to use this task?

You can use this task either with a full Anaconda installation or a Miniconda installation. If using a self-hosted agent, you must add the `conda` executable to `PATH`. Alternatively, you can set the `CONDA` environment variable to the root of the Conda installation -- that is, the directory you specify as the "prefix" when installing Conda.

Package: npm

7/19/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

Install and publish npm packages

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Install npm packages

Demands

npm

YAML snippet

```
# npm
# Install and publish npm packages, or run an npm command. Supports npmjs.com and authenticated registries like Package Management.
- task: Npm@1
  inputs:
    #command: 'install' # Options: install, publish, custom
    #workingDir: # Optional
    #verbose: # Optional
    #customCommand: # Required when command == Custom
    #customRegistry: 'useNpmrc' # Optional. Options: useNpmrc, useFeed
    #customFeed: # Required when customRegistry == UseFeed
    #customEndpoint: # Optional
    #publishRegistry: 'useExternalRegistry' # Optional. Options: useExternalRegistry, useFeed
    #publishFeed: # Required when publishRegistry == UseFeed
    #publishEndpoint: # Required when publishRegistry == UseExternalRegistry
```

Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select `install` here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
CUSTOM REGISTRIES AND AUTHENTICATION	

ARGUMENT	DESCRIPTION
Registries to use	*Leave this section blank to use packages from npmjs directly.* Otherwise, select one of these options: **Registries in my .npmrc:** • Select this option to use feeds specified in a .npmrc file you've checked into source control. • Credentials for registries outside this organization/collection can be used to inject credentials you've provided as an npm service connection into your .npmrc as the build runs. **Use packages from this VSTS/TFS registry:** • Select this option to use one Package Management feed in the same organization/collection as the build.
ADVANCED	
Verbose logging	Enables verbose logging.
CONTROL OPTIONS	

Publish npm packages

Demands

npm

Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select `publish` here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
DESTINATION REGISTRY AND AUTHENTICATION	
Registry location	• **Registry I select here** publishes to a Package Management registry in the same organization/collection as the build. After you select this option, select the target registry from the dropdown. • **External npm registry (including other organizations/collections)** publishes to an external server such as npm, MyGet, or a Package Management feed in another VSTS organization or TFS collection. After you select this option, create and select an npm service connection.
ADVANCED	
Verbose logging	Enables verbose logging.
CONTROL OPTIONS	

Custom npm command

Demands

npm

Arguments

ARGUMENT	DESCRIPTION
Command	npm command to run. Select `custom` here.
Working folder with package.json	Path to the folder containing the target package.json and .npmrc files. Select the folder, not the file e.g. "/packages/mypackage".
Command and arguments	The custom command and arguments you wish to be executed. If your arguments contain double quotes (\"), escape them with a slash (\\), and surround the escaped string with double quotes (\"). Example: to run `npm run myTask -- --users='{"foo":"bar"}'`, provide this input: `run myTask -- --users="{\"foo\":\"bar\"}"`.
CUSTOM REGISTRIES AND AUTHENTICATION	
Registries to use	*Leave this section blank to use packages from npmjs directly.* Otherwise, select one of these options: **Registries in my .npmrc:** • Select this option to use feeds specified in a .npmrc file you've checked into source control. • Credentials for registries outside this organization/collection can be used to inject credentials you've provided as an npm service connection into your .npmrc as the build runs. **Use packages from this VSTS/TFS registry:** • Select this option to use one Package Management feed in the same organization/collection as the build.
CONTROL OPTIONS	

Examples

Build: Gulp

Build your Node.js app with Gulp

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn npm commands and arguments?

npm docs

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 Authenticate (for task runners)

7/9/2018 • 2 minutes to read • Edit Online

Don't use this task if you're also using the npm task. Provides npm credentials to an .npmrc file in your repository for the scope of the build. This enables npm task runners like Gulp and Grunt to authenticate with private registries.

YAML snippet

```
# npm Authenticate (for task runners)
# Don't use this task if you're also using the npm task. Provides npm credentials to an .npmrc file in your
repository for the scope of the build. This enables npm task runners like Gulp and Grunt to authenticate with
private registries.
- task: npmAuthenticate@0
  inputs:
    #workingFile: # Optional
    #customEndpoint: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
.npmrc file to authenticate	(Optional) Path to the .npmrc file that specifies the registries you want to work with. Select the file, not the folder e.g. "/packages/mypackage.npmrc".
Credentials for registries outside this account/collection	(Optional) Credentials to use for external registries located in the project's .npmrc. For registries in this account/collection, leave this blank; the build's credentials are used automatically.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Package: NuGet

7/19/2018 • 11 minutes to read • Edit Online

Version 2.* | Other versions

VSTS | TFS 2018

Install and update NuGet package dependencies, or package and publish NuGet packages.

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

If your code depends on NuGet packages, make sure to add this step before your Visual Studio Build step. Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

> **TIP**
> Looking for help to get started? See the how-to's for restoring and publishing packages.

> **TIP**
> This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the Tool Installer.

> **NOTE**
> Using or creating .NET Core or .NET Standard packages? Use the .NET Core task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

YAML snippet

```
# NuGet
# Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like
Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard
apps, use the .NET Core task.
- task: NuGetCommand@2
  inputs:
    #command: 'restore' # Options: restore, pack, push, custom
    #restoreSolution: '**/*.sln' # Required when command == Restore
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    #disableParallelProcessing: false
    restoreDirectory:
    #verbosityRestore: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPush:
'$(Build.ArtifactStagingDirectory)/**/*.nupkg;!$(Build.ArtifactStagingDirectory)/**/*.symbols.nupkg' # Required when
command == Push
    #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
    #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
    #allowPackageConflicts: # Optional
    #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
    #verbosityPush: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPack: '**/*.csproj' # Required when command == Pack
    #configuration: '$(BuildConfiguration)' # Optional
    #packDestination: '$(Build.ArtifactStagingDirectory)' # Optional
    #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
    #includeReferencedProjects: false # Optional
    #versionEnvVar: # Required when versioningScheme == ByEnvVar
    #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
    #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #packTimezone: 'utc' # Required when versioningScheme == ByPrereleaseNumber# Options: utc, local
    #includeSymbols: false # Optional
    #toolPackage: # Optional
    #buildProperties: # Optional
    #basePath: # Optional
    #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
    #arguments: # Required when command == Custom
```

Versioning schemes

For **byPrereleaseNumber**, the version will be set to whatever you choose for major, minor, and patch, plus the date and time in the format `yyyymmdd-hhmmss`.

For **byEnvVar**, the version will be set as whatever environment variable, e.g. $(MyVersion), you provide. Make sure the environment variable is set to a proper SemVer e.g. `1.2.3` or `1.2.3-beta1`.

For **byBuildNumber**, the version will be set to the build number, ensure that your build number is a proper SemVer e.g. `1.0.$(Rev:r)`.

Restore NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the **Solution** argument in your Visual Studio Build step and paste it here, or create a link using the Link button in the information panel.

ARGUMENT	DESCRIPTION
FEEDS AND AUTHENTICATION	
Feeds to use	**Feed(s) I select here:** • Select this option to use NuGet.org and/or one Package Management feed in the same organization/collection as the build. **Feeds in my NuGet.config:** • Select this option to use feeds specified in a NuGet.config file you've checked into source control. • Credentials for feeds outside this organization/collection can be used to inject credentials you've provided as a NuGet service connection into your NuGet.config as the build runs. • See the walkthrough for help using packages from feeds in multiple VSTS organizations.
ADVANCED	
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```
`-- ConsoleApplication1
    |-- ConsoleApplication1.sln
    |-- NuGet.config
    `-- ConsoleApplication1
        |-- ConsoleApplication1.csproj
```

Build steps

Package: NuGet	Install your NuGet package dependencies. • Path to solution, packages.config, or project.json: `**/*.sln` • Feeds to use: Feeds in my NuGet.config • Path to NuGet.config: `ConsoleApplication1/NuGet.config`
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Restore NuGet Packages: **(Important)** Make sure this option is cleared.

Pack NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	Specify .csproj files (for example, `**\*.csproj`) for simple projects. In this case: • The packager compiles the .csproj files for packaging. • You must specify **Configuration to Package** (see below). • You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as *id* and *$description$*. Specify .nuspec files (for example, `**\*.nuspec`) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case: • The packager does not compile the .csproj files for packaging. • Each project is packaged only if it has a .nuspec file checked in. • The packager does not replace tokens in the .nuspec file (except the `<version/>` element, see **Use build number to version package**, below). You must supply values for elements such as `<id/>` and `<description/>`. The most common way to do this is to hardcode the values in the .nuspec file. To package a single file, click the **...** button and select the file. To package multiple files, use file matching patterns. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains `-:`, use `!` instead.
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: `Release` or `$(BuildConfiguration)`
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as `$(Build.ArtifactStagingDirectory)` If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	This blog post provides an overview of the automatic package versioning available here.
ADVANCED	
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace `<description>$description$</description>` in the .nuspec file this way: `Description="This is a great package"` Using this argument is equivalent to supplying properties from nuget pack with the `-Properties` option.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	Specify the packages you want to publish. • Default value: `$(Build.ArtifactStagingDirectory)/*.nupkg` • To publish a single package, click the **...** button and select the file. • Use file matching patterns to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains `-:`, use `!` instead. • Use variables to specify directories. For example, if you specified `$(Build.ArtifactStagingDirectory)\` as the **package folder** in the pack step above, you could specify `$(Build.ArtifactStagingDirectory)\**\*.nupkg` here.

Target feed location	• **This organization/collection** publishes to a Package Management feed in the same organization/collection as the build. After you select this option, select the target feed from the dropdown.
	  ◦ "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.
	• **External NuGet server (including other organizations/collections)** publishes to an external server such as NuGet, MyGet, or a Package Management feed in another VSTS organization or TFS collection. After you select this option, you create and select a NuGet service connection.

ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Publishing symbols

When you push packages to 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

YAML snippet

```
# NuGet Command
# Deprecated: use the “NuGet” task instead. It works with the new Tool Installer framework so you can easily use new versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this organization/collection, and uses NuGet 4 by default.
- task: NuGet@0
  inputs:
    command:
    arguments:
```

Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

End-to-end example

You want to package and publish some projects in a C# class library to your 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
`$(BuildConfiguration)`	`release`
`$(BuildPlatform)`	`any cpu`

Options

SETTING	VALUE
Build number format	`$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.`

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:

Package: NuGet	Install your NuGet package dependencies. • Path to solution, packages.config, or project.json: `**/*.sln` • Feeds to use: Feeds I select here • Use packages from NuGet.org: Checked
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
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 organization/collection • Target feed: Select your feed

Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described above.
2. If you haven't already, register with NuGet.org.
3. Use the steps in the previous section, but substitute the final step for the step shown here.

Package: NuGet	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 connection with your NuGet.org ApiKey and select it here

Task versions

Task: NuGet (formerly NuGet Restore at 1.*, NuGet Installer at 0.*)

TASK VERSION	VSTS	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018

TASK VERSION	VSTS	TFS
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

Open source

These tasks are open source on GitHub. Feedback and contributions are welcome.

Q & A

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project-the location of its packages-is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add 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 Seting up a solution to get dependencies

What other kinds of apps can I build?

Build your app

What other kinds of build tasks are available?

Specify your build tasks

How do we protect our codebase from build breaks?

- Git: Improve code quality with branch policies with an option to require that code builds before it can be merged to

a branch. This option is not available for GitHub repos.

- TFVC: Use gated check-in.

How do I modify other parts of my build pipeline?

- Specify your build tasks to run tests, scripts, and a wide range of other processes.
- Specify build options such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- Specify the repository to pick the source of the build and modify options such as how the agent workspace is cleaned.
- Set build triggers to modify how your CI builds run and to specify scheduled builds.
- Specify build retention policies to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent jobs. See Concurrent build and release jobs in VSTS.

How do I see what has changed in my build pipeline?

View the change history of your build pipeline

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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: PyPI Publisher

7/9/2018 • 2 minutes to read • Edit Online

Create and upload an sdist or wheel to a PyPI-compatible index using Twine.

This task builds an sdist package by running `python setup.py sdist` using the Python instance in `PATH`. It can optionally build a universal wheel in addition to the sdist. Then, it will upload the package to a PyPI index using `twine`. The task will install the `wheel` and `twine` packages with `python -m pip install --user`.

Demands

None

Prerequisites

A generic service connection for a PyPI index.

TIP

To configure a new generic service connection, go to Settings -> Services -> New service connection -> Generic.

- **Connection Name**: A friendly connection name of your choice
- **Server URL**: PyPI package server (for example: https://upload.pypi.org/legacy/)
- **User name**: username for your PyPI account
- **Password/Token Key**: password for your PyPI account

YAML snippet

```
# PyPI Publisher
# Create and upload an sdist or wheel to a PyPI-compatible index using Twine.
- task: PyPIPublisher@0
  inputs:
    pypiConnection:
    packageDirectory:
    #alsoPublishWheel: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
PyPI connection	A generic service connection for connecting to the package index.
Python package directory	The directory of the Python package to be created and published, where setup.py is present.
Also publish a wheel	Select whether to create and publish a universal wheel package (platform independent) in addition to an sdist package.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Package: NuGet Installer version 0.*

5/31/2018 • 2 minutes 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 task before your Visual Studio Build task. Also make sure to clear the deprecated **Restore NuGetPackages** checkbox in that task.

Arguments

ARGUMENT	DESCRIPTION
Path to Solution	Copy the value from the **Solution** argument in your Visual Studio Build task and paste it here.
Path to NuGet.config	If you are using a package source other than NuGet.org, you must check in a NuGet.config file and specify the path to it here.
Disable local cache	Equivalent to nuget restore with the `-NoCache` option.
NuGet Arguments	Additional arguments passed to nuget restore.
ADVANCED	
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your own strategy to handle authentication.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```
`-- ConsoleApplication1
    |-- ConsoleApplication1.sln
    |-- NuGet.config
    `-- ConsoleApplication1
        |-- ConsoleApplication1.csproj
```

Build tasks

Package: NuGet Installer	Install your NuGet package dependencies. • Path to Solution: `**\*.sln` • Path to NuGet.config: `ConsoleApplication1/NuGet.config`
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Restore NuGet Packages: **(Important)** Make sure this option is cleared.

Package: NuGet

7/19/2018 • 11 minutes to read • Edit Online

Version 2.* | Other versions

VSTS | TFS 2018

Install and update NuGet package dependencies, or package and publish NuGet packages.

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

If your code depends on NuGet packages, make sure to add this step before your Visual Studio Build step. Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

> **TIP**
> Looking for help to get started? See the how-to's for restoring and publishing packages.

> **TIP**
> This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the Tool Installer.

> **NOTE**
> Using or creating .NET Core or .NET Standard packages? Use the .NET Core task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

YAML snippet

```
# NuGet
# Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like
Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard
apps, use the .NET Core task.
- task: NuGetCommand@2
  inputs:
    #command: 'restore' # Options: restore, pack, push, custom
    #restoreSolution: '**/*.sln' # Required when command == Restore
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    #disableParallelProcessing: false
    restoreDirectory:
    #verbosityRestore: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPush:
'$(Build.ArtifactStagingDirectory)/**/*.nupkg;!$(Build.ArtifactStagingDirectory)/**/*.symbols.nupkg' # Required when
command == Push
    #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
    #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
    #allowPackageConflicts: # Optional
    #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
    #verbosityPush: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPack: '**/*.csproj' # Required when command == Pack
    #configuration: '$(BuildConfiguration)' # Optional
    #packDestination: '$(Build.ArtifactStagingDirectory)' # Optional
    #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
    #includeReferencedProjects: false # Optional
    #versionEnvVar: # Required when versioningScheme == ByEnvVar
    #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
    #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #packTimezone: 'utc' # Required when versioningScheme == ByPrereleaseNumber# Options: utc, local
    #includeSymbols: false # Optional
    #toolPackage: # Optional
    #buildProperties: # Optional
    #basePath: # Optional
    #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
    #arguments: # Required when command == Custom
```

Versioning schemes

For **byPrereleaseNumber**, the version will be set to whatever you choose for major, minor, and patch, plus the date and time in the format `yyyymmdd-hhmmss`.

For **byEnvVar**, the version will be set as whatever environment variable, e.g. $(MyVersion), you provide. Make sure the environment variable is set to a proper SemVer e.g. `1.2.3` or `1.2.3-beta1`.

For **byBuildNumber**, the version will be set to the build number, ensure that your build number is a proper SemVer e.g. `1.0.$(Rev:r)`.

Restore NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the **Solution** argument in your Visual Studio Build step and paste it here, or create a link using the Link button in the information panel.

ARGUMENT	DESCRIPTION
FEEDS AND AUTHENTICATION	
Feeds to use	**Feed(s) I select here:** • Select this option to use NuGet.org and/or one Package Management feed in the same organization/collection as the build. **Feeds in my NuGet.config:** • Select this option to use feeds specified in a NuGet.config file you've checked into source control. • Credentials for feeds outside this organization/collection can be used to inject credentials you've provided as a NuGet service connection into your NuGet.config as the build runs. • See the walkthrough for help using packages from feeds in multiple VSTS organizations.
ADVANCED	
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```
`-- ConsoleApplication1
    |-- ConsoleApplication1.sln
    |-- NuGet.config
    `-- ConsoleApplication1
        |-- ConsoleApplication1.csproj
```

Build steps

Package: NuGet	Install your NuGet package dependencies. • Path to solution, packages.config, or project.json: `**/*.sln` • Feeds to use: Feeds in my NuGet.config • Path to NuGet.config: `ConsoleApplication1/NuGet.config`
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Restore NuGet Packages: **(Important)** Make sure this option is cleared.

Pack NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	Specify .csproj files (for example, `**\*.csproj`) for simple projects. In this case: • The packager compiles the .csproj files for packaging. • You must specify **Configuration to Package** (see below). • You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as *id* and *$description$*. Specify .nuspec files (for example, `**\*.nuspec`) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case: • The packager does not compile the .csproj files for packaging. • Each project is packaged only if it has a .nuspec file checked in. • The packager does not replace tokens in the .nuspec file (except the `<version/>` element, see **Use build number to version package**, below). You must supply values for elements such as `<id/>` and `<description/>`. The most common way to do this is to hardcode the values in the .nuspec file. To package a single file, click the **...** button and select the file. To package multiple files, use file matching patterns. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains `-:`, use `!` instead.
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: `Release` or `$(BuildConfiguration)`
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as `$(Build.ArtifactStagingDirectory)` If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	This blog post provides an overview of the automatic package versioning available here.
ADVANCED	
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace `<description>$description$</description>` in the .nuspec file this way: `Description="This is a great package"` Using this argument is equivalent to supplying properties from nuget pack with the `-Properties` option.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	Specify the packages you want to publish. • Default value: `$(Build.ArtifactStagingDirectory)/*.nupkg` • To publish a single package, click the **...** button and select the file. • Use file matching patterns to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains `-:`, use `!` instead. • Use variables to specify directories. For example, if you specified `$(Build.ArtifactStagingDirectory)\` as the **package folder** in the pack step above, you could specify `$(Build.ArtifactStagingDirectory)\**\*.nupkg` here.

Target feed location	• **This organization/collection** publishes to a Package Management feed in the same organization/collection as the build. After you select this option, select the target feed from the dropdown.
	  ◦ "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.
	• **External NuGet server (including other organizations/collections)** publishes to an external server such as NuGet, MyGet, or a Package Management feed in another VSTS organization or TFS collection. After you select this option, you create and select a NuGet service connection.

ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Publishing symbols

When you push packages to 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

YAML snippet

```
# NuGet Command
# Deprecated: use the “NuGet” task instead. It works with the new Tool Installer framework so you can easily use new versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this organization/collection, and uses NuGet 4 by default.
- task: NuGet@0
  inputs:
    command:
    arguments:
```

Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

End-to-end example

You want to package and publish some projects in a C# class library to your 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
`$(BuildConfiguration)`	`release`
`$(BuildPlatform)`	`any cpu`

Options

SETTING	VALUE
Build number format	`$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:.`

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:

Package: NuGet	Install your NuGet package dependencies. • Path to solution, packages.config, or project.json: `**/*.sln` • Feeds to use: Feeds I select here • Use packages from NuGet.org: Checked
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
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 organization/collection • Target feed: Select your feed

Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described above.
2. If you haven't already, register with NuGet.org.
3. Use the steps in the previous section, but substitute the final step for the step shown here.

Package: NuGet	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 connection with your NuGet.org ApiKey and select it here

Task versions

Task: NuGet (formerly NuGet Restore at 1.*, NuGet Installer at 0.*)

TASK VERSION	VSTS	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018

TASK VERSION	VSTS	TFS
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

Open source

These tasks are open source on GitHub. Feedback and contributions are welcome.

Q & A

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project-the location of its packages-is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add 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 Seting up a solution to get dependencies

What other kinds of apps can I build?

Build your app

What other kinds of build tasks are available?

Specify your build tasks

How do we protect our codebase from build breaks?

- Git: Improve code quality with branch policies with an option to require that code builds before it can be merged to

a branch. This option is not available for GitHub repos.

- TFVC: Use gated check-in.

How do I modify other parts of my build pipeline?

- Specify your build tasks to run tests, scripts, and a wide range of other processes.
- Specify build options such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- Specify the repository to pick the source of the build and modify options such as how the agent workspace is cleaned.
- Set build triggers to modify how your CI builds run and to specify scheduled builds.
- Specify build retention policies to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent jobs. See Concurrent build and release jobs in VSTS.

How do I see what has changed in my build pipeline?

View the change history of your build pipeline

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.*

7/9/2018 • 5 minutes 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.

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Demands

None

YAML snippet

```
# NuGet Packager
# Deprecated: use the “NuGet” task instead. It works with the new Tool Installer framework so you can easily use new versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this organization/collection, and uses NuGet 4 by default.
- task: NuGetPackager@0
  inputs:
    #searchPattern: '**\*.csproj'
    #outputdir: # Optional
    #includeReferencedProjects: false # Optional
    #versionByBuild: 'false' # Options: false, byPrereleaseNumber, byEnvVar, true
    #versionEnvVar: # Required when versionByBuild == ByEnvVar
    #requestedMajorVersion: '1' # Required when versionByBuild == ByPrereleaseNumber
    #requestedMinorVersion: '0' # Required when versionByBuild == ByPrereleaseNumber
    #requestedPatchVersion: '0' # Required when versionByBuild == ByPrereleaseNumber
    #configurationToPack: '$(BuildConfiguration)' # Optional
    #buildProperties: # Optional
    #nuGetAdditionalArgs: # Optional
    #nuGetPath: # Optional
```

Arguments

ARGUMENT	DESCRIPTION

ARGUMENT	DESCRIPTION
Path/Pattern to nuspec files	Specify .csproj files (for example, `**\*.csproj`) for simple projects. In this case: • The packager compiles the .csproj files for packaging. • You must specify **Configuration to Package** (see below). • You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as *id* and *$description$*. Specify .nuspec files (for example, `**\*.nuspec`) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case: • The packager does not compile the .csproj files for packaging. • Each project is packaged only if it has a .nuspec file checked in. • The packager does not replace tokens in the .nuspec file (except the `<version/>` element, see **Use build number to version package**, below). You must supply values for elements such as `<id/>` and `<description/>`. The most common way to do this is to hardcode the values in the .nuspec file. To package a single file, click the **...** button and select the file. To package multiple files, use single-folder wildcards (`*`) and recursive wildcards (`**`). For example, specify `**\*.csproj` to package all .csproj files in all subdirectories in the repo. You can use multiple patterns separated by a semicolon to create more complex queries. You can negate a pattern by prefixing it with "-:". For example, specify `**\*.csproj;-:**\*Tests.csproj` to package all .csproj files except those ending in 'Tests' in all subdirectories in the repo.
Use build number to version package	Select if you want to use the build number to version your package. If you select this option, for the pipeline options, set the **build number format** to something like `$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:` The build number format must be `{some_characters}_0.0.0.0`. The characters and the underscore character are omitted from the output. The version number at the end must be a unique number in a format such as `0.0.0.0` that is higher than the last published number. The version number is passed to nuget pack with the `-Version` option. Versions are shown prominently on NuGet servers. For example they are listed on the Visual Studio Team Services feeds page and on the NuGet.org package page.
Package Folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as `$(Build.StagingDirectory)\packages` If you leave it empty, the package will be created in the same directory that contains the .csproj or .nuspec file.
ADVANCED	

ARGUMENT	DESCRIPTION
Configuration to Package	If you are packaging a .csproj file, you must specify a configuration that you are building and that you want to package. For example: `Release`
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace `<description>$description$</description>` in the .nuspec file this way: `Description="This is a great package"` Using this argument is equivalent to supplying properties from nuget pack with the `-Properties` option.
NuGet Arguments	(Optional) Additional arguments passed nuget pack.
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your own strategy to handle authentication.
CONTROL OPTIONS	

Examples

You want to package and publish some projects in a C# class library to your 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
`$(BuildConfiguration)`	`release`
`$(BuildPlatform)`	`any cpu`

Options

SETTING	VALUE
Build number format	`$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:`

Publish to VSTS

Make sure you've prepared the build as described above.

Create the feed

See Create a feed.

Build tasks

Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
Package: NuGet Packager	Package your projects. • Path/Pattern to nuspec files: `**\*.csproj` • Use Build number to version package: Selected • Advanced, Configuration to Package: `Release`
Package: NuGet Publisher	Publish your packages to VSTS. • Path/Pattern to nupkg: `**\*.nupkg` • Feed type: Internal NuGet Feed • Internal feed URL: See Find your NuGet package source URL.

Publish to NuGet.org

Make sure you've prepared the build as described above.

Register with NuGet.org

If you haven't already, register with NuGet.org.

Build tasks

Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
Package: NuGet Packager	Package your projects. • Path/Pattern to nuspec files: `**\*.csproj` • Use Build number to version package: Selected • Advanced, Configuration to Package: `Release`

Package: NuGet Publisher	Publish your packages to NuGet.org.
	• Path/Pattern to nupkg: `**\*.nupkg`
	• Feed type: External NuGet Feed
	• NuGet Server Endpoint:
	1. Click "New service connection", and then click Generic.
	2. On the Add New Generic Connection dialog box:
	  ○ Connection Name: `NuGet`
	  ○ Server URL: `https://nuget.org/`
	  ○ User name: `{your-name}`
	  ○ Password/Token Key: Paste API Key from your NuGet account.

Package: NuGet Publisher version 0.*

7/9/2018 • 3 minutes 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

YAML snippet

```
# NuGet Publisher
# Deprecated: use the “NuGet” task instead. It works with the new Tool Installer framework so you can easily use new versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this organization/collection, and uses NuGet 4 by default.
- task: NuGetPublisher@0
  inputs:
    #searchPattern: '**/*.nupkg;-:**/packages/**/*.nupkg;-:**/*.symbols.nupkg'
    #nuGetFeedType: 'external' # Options: external, internal
    #connectedServiceName: # Required when nuGetFeedType == External
    #feedName: # Required when nuGetFeedType == Internal
    #nuGetAdditionalArgs: # Optional
    #verbosity: '-' # Options: -, quiet, normal, detailed
    #nuGetVersion: '3.3.0' # Options: 3.3.0, 3.5.0.1829, 4.0.0.2283, custom
    #nuGetPath: # Optional
    #continueOnEmptyNupkgMatch: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Path/Pattern to nupkg	Specify the packages you want to publish. • Default value: `**\*.nupkg;-:**\packages\**\*.nupkg` • To publish a single package, click the **...** button and select the file. • Use single-folder wildcards (`*`) and recursive wildcards (`**`) to publish multiple packages. • Use variables to specify directories. For example, if you specified `$(Build.StagingDirectory)\packages` as the **package folder** in the NuGet Packager task, you could specify `$(Build.StagingDirectory)\packages\**\*.nupkg` here.
Feed type	• **External NuGetFeed** publishes to an external server such as NuGet or MyGet. After you select this option, you create and select a **NuGet server endpoint**. • **Internal NuGet Feed** publishes to an internal or Visual Studio Team Services feed. After you select this option, you specify the **internal feed URL**.
ADVANCED	

ARGUMENT	DESCRIPTION
NuGet Arguments	(Optional) Additional arguments passed to nuget push.
Path to NuGet.exe	(Optional) Path to your own instance of NuGet.exe. If you specify this argument, you must have your own strategy to handle authentication.
CONTROL OPTIONS	

Examples

You want to package and publish some projects in a C# class library to your 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
`$(BuildConfiguration)`	`release`
`$(BuildPlatform)`	`any cpu`

Options

SETTING	VALUE
Build number format	`$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:`

Publish to VSTS

Make sure you've prepared the build as described above.

Create the feed

See Create a feed.

Build tasks

Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
Package: NuGet Packager	Package your projects. • Path/Pattern to nuspec files: `**\*.csproj` • Use Build number to version package: Selected • Advanced, Configuration to Package: `Release`
Package: NuGet Publisher	Publish your packages to VSTS. • Path/Pattern to nupkg: `**\*.nupkg` • Feed type: Internal NuGet Feed • Internal feed URL: See Find your NuGet package source URL.

Publish to NuGet.org

Make sure you've prepared the build as described above.

Register with NuGet.org

If you haven't already, register with NuGet.org.

Build tasks

Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
Package: NuGet Packager	Package your projects. • Path/Pattern to nuspec files: `**\*.csproj` • Use Build number to version package: Selected • Advanced, Configuration to Package: `Release`

Package: NuGet Publisher	Publish your packages to NuGet.org.
	• Path/Pattern to nupkg: `**\*.nupkg`
	• Feed type: External NuGet Feed
	• NuGet Server Endpoint:
	1. Click "New service connection", and then click Generic.
	2. On the Add New Generic Connection dialog box:
	  ○ Connection Name: `NuGet`
	  ○ Server URL: `https://nuget.org/`
	  ○ User name: `{your-name}`
	  ○ Password/Token Key: Paste API Key from your NuGet account.

Package: NuGet

7/19/2018 • 11 minutes to read • Edit Online

Version 2.* | Other versions

VSTS | TFS 2018

Install and update NuGet package dependencies, or package and publish NuGet packages.

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

If your code depends on NuGet packages, make sure to add this step before your Visual Studio Build step. Also make sure to clear the deprecated **Restore NuGet Packages** checkbox in that step.

> **TIP**
> Looking for help to get started? See the how-to's for restoring and publishing packages.

> **TIP**
> This version of the NuGet task uses NuGet 4.1.0 by default. To select a different version of NuGet, use the Tool Installer.

> **NOTE**
> Using or creating .NET Core or .NET Standard packages? Use the .NET Core task, which has full support for all package scenarios currently supported by dotnet, including restore, pack, and nuget push.

YAML snippet

```
# NuGet
# Restore, pack, or push NuGet packages, or run a NuGet command. Supports NuGet.org and authenticated feeds like
Package Management and MyGet. Uses NuGet.exe and works with .NET Framework apps. For .NET Core and .NET Standard
apps, use the .NET Core task.
- task: NuGetCommand@2
  inputs:
    #command: 'restore' # Options: restore, pack, push, custom
    #restoreSolution: '**/*.sln' # Required when command == Restore
    #feedsToUse: 'select' # Options: select, config
    #vstsFeed: # Required when feedsToUse == Select
    #includeNuGetOrg: true # Required when feedsToUse == Select
    #nugetConfigPath: # Required when feedsToUse == Config
    #externalFeedCredentials: # Optional
    #noCache: false
    #disableParallelProcessing: false
    restoreDirectory:
    #verbosityRestore: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPush:
'$(Build.ArtifactStagingDirectory)/**/*.nupkg;!$(Build.ArtifactStagingDirectory)/**/*.symbols.nupkg' # Required when
command == Push
    #nuGetFeedType: 'internal' # Required when command == Push# Options: internal, external
    #publishVstsFeed: # Required when command == Push && NuGetFeedType == Internal
    #allowPackageConflicts: # Optional
    #publishFeedCredentials: # Required when command == Push && NuGetFeedType == External
    #verbosityPush: 'Detailed' # Options: quiet, normal, detailed
    #packagesToPack: '**/*.csproj' # Required when command == Pack
    #configuration: '$(BuildConfiguration)' # Optional
    #packDestination: '$(Build.ArtifactStagingDirectory)' # Optional
    #versioningScheme: 'off' # Options: off, byPrereleaseNumber, byEnvVar, byBuildNumber
    #includeReferencedProjects: false # Optional
    #versionEnvVar: # Required when versioningScheme == ByEnvVar
    #majorVersion: '1' # Required when versioningScheme == ByPrereleaseNumber
    #minorVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #patchVersion: '0' # Required when versioningScheme == ByPrereleaseNumber
    #packTimezone: 'utc' # Required when versioningScheme == ByPrereleaseNumber# Options: utc, local
    #includeSymbols: false # Optional
    #toolPackage: # Optional
    #buildProperties: # Optional
    #basePath: # Optional
    #verbosityPack: 'Detailed' # Options: quiet, normal, detailed
    #arguments: # Required when command == Custom
```

Versioning schemes

For **byPrereleaseNumber**, the version will be set to whatever you choose for major, minor, and patch, plus the date and time in the format `yyyymmdd-hhmmss`.

For **byEnvVar**, the version will be set as whatever environment variable, e.g. $(MyVersion), you provide. Make sure the environment variable is set to a proper SemVer e.g. `1.2.3` or `1.2.3-beta1`.

For **byBuildNumber**, the version will be set to the build number, ensure that your build number is a proper SemVer e.g. `1.0.$(Rev:r)`.

Restore NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to solution, packages.config, or project.json	Copy the **Solution** argument in your Visual Studio Build step and paste it here, or create a link using the Link button in the information panel.

ARGUMENT	DESCRIPTION
FEEDS AND AUTHENTICATION	
Feeds to use	**Feed(s) I select here:** • Select this option to use NuGet.org and/or one Package Management feed in the same organization/collection as the build. **Feeds in my NuGet.config:** • Select this option to use feeds specified in a NuGet.config file you've checked into source control. • Credentials for feeds outside this organization/collection can be used to inject credentials you've provided as a NuGet service connection into your NuGet.config as the build runs. • See the walkthrough for help using packages from feeds in multiple VSTS organizations.
ADVANCED	
Disable local cache	Prevents NuGet from using packages from local machine caches.
Destination directory	Specifies the folder in which packages are installed. If no folder is specified, packages are restored into a packages/ folder alongside the selected solution, packages.config, or project.json.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Examples

Install NuGet dependencies

You're building a Visual Studio solution that depends on a NuGet feed.

```
`-- ConsoleApplication1
    |-- ConsoleApplication1.sln
    |-- NuGet.config
    `-- ConsoleApplication1
        |-- ConsoleApplication1.csproj
```

Build steps

Package: NuGet	Install your NuGet package dependencies. • Path to solution, packages.config, or project.json: `**/*.sln` • Feeds to use: Feeds in my NuGet.config • Path to NuGet.config: `ConsoleApplication1/NuGet.config`
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Restore NuGet Packages: **(Important)** Make sure this option is cleared.

Pack NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to csproj or nuspec file(s) to pack	Specify .csproj files (for example, `**\*.csproj`) for simple projects. In this case: • The packager compiles the .csproj files for packaging. • You must specify **Configuration to Package** (see below). • You do not have to check in a .nuspec file. If you do check one in, the packager honors its settings and replaces tokens such as *id* and *$description$*. Specify .nuspec files (for example, `**\*.nuspec`) for more complex projects, such as multi-platform scenarios in which you need to compile and package in separate steps. In this case: • The packager does not compile the .csproj files for packaging. • Each project is packaged only if it has a .nuspec file checked in. • The packager does not replace tokens in the .nuspec file (except the `<version/>` element, see **Use build number to version package**, below). You must supply values for elements such as `<id/>` and `<description/>`. The most common way to do this is to hardcode the values in the .nuspec file. To package a single file, click the **...** button and select the file. To package multiple files, use file matching patterns. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains `-:`, use `!` instead.
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: `Release` or `$(BuildConfiguration)`
Package folder	(Optional) Specify the folder where you want to put the packages. You can use a variable such as `$(Build.ArtifactStagingDirectory)` If you leave it empty, the package will be created in the root of your source tree.
PACK OPTIONS	
Automatic package versioning	This blog post provides an overview of the automatic package versioning available here.
ADVANCED	
Additional build properties	Semicolon delimited list of properties used to build the package. For example, you could replace `<description>$description$</description>` in the .nuspec file this way: `Description="This is a great package"` Using this argument is equivalent to supplying properties from nuget pack with the `-Properties` option.
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Push NuGet packages

Demands

None

Arguments

ARGUMENT	DESCRIPTION
Path to NuGet package(s) to publish	Specify the packages you want to publish. • Default value: `$(Build.ArtifactStagingDirectory)/*.nupkg` • To publish a single package, click the **...** button and select the file. • Use file matching patterns to publish multiple packages. Note that these patterns were updated in version 2 of the NuGet task; if you have a pattern that contains `-:`, use `!` instead. • Use variables to specify directories. For example, if you specified `$(Build.ArtifactStagingDirectory)\` as the **package folder** in the pack step above, you could specify `$(Build.ArtifactStagingDirectory)\**\*.nupkg` here.

Target feed location	• **This organization/collection** publishes to a Package Management feed in the same organization/collection as the build. After you select this option, select the target feed from the dropdown.
	  ◦ "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.
	• **External NuGet server (including other organizations/collections)** publishes to an external server such as NuGet, MyGet, or a Package Management feed in another VSTS organization or TFS collection. After you select this option, you create and select a NuGet service connection.

ADVANCED	
Verbosity	Specifies the amount of detail displayed in the output.
CONTROL OPTIONS	

Publishing symbols

When you push packages to 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

YAML snippet

```
# NuGet Command
# Deprecated: use the “NuGet” task instead. It works with the new Tool Installer framework so you can easily use new versions of NuGet without waiting for a task update, provides better support for authenticated feeds outside this organization/collection, and uses NuGet 4 by default.
- task: NuGet@0
  inputs:
    command:
    arguments:
```

Arguments

ARGUMENT	DESCRIPTION
Command and arguments	NuGet command and arguments you want to pass as your custom command.

End-to-end example

You want to package and publish some projects in a C# class library to your 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
`$(BuildConfiguration)`	`release`
`$(BuildPlatform)`	`any cpu`

Options

SETTING	VALUE
Build number format	`$(BuildDefinitionName)_$(Year:yyyy).$(Month).$(DayOfMonth)$(Rev:`

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:

Package: NuGet	Install your NuGet package dependencies. • Path to solution, packages.config, or project.json: `**/*.sln` • Feeds to use: Feeds I select here • Use packages from NuGet.org: Checked
Build: Visual Studio Build	Build your solution. • Solution: `**\*.sln` • Platform: `$(BuildPlatform)` • Configuration: `$(BuildConfiguration)`
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 organization/collection • Target feed: Select your feed

Option 2: publish to NuGet.org

1. Make sure you've prepared the build as described above.
2. If you haven't already, register with NuGet.org.
3. Use the steps in the previous section, but substitute the final step for the step shown here.

Package: NuGet	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 connection with your NuGet.org ApiKey and select it here

Task versions

Task: NuGet (formerly NuGet Restore at 1.*, NuGet Installer at 0.*)

TASK VERSION	VSTS	TFS
2.*	Available	Appeared in 2018
1.*	Deprecated but available	Appeared in 2017 Update 2, deprecated in 2018

TASK VERSION	VSTS	TFS
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

Open source

These tasks are open source on GitHub. Feedback and contributions are welcome.

Q & A

Why should I check in a NuGet.Config?

Checking a NuGet.Config into source control ensures that a key piece of information needed to build your project-the location of its packages-is available to every developer that checks out your code.

However, for situations where a team of developers works on a large range of projects, it's also possible to add 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 Seting up a solution to get dependencies

What other kinds of apps can I build?

Build your app

What other kinds of build tasks are available?

Specify your build tasks

How do we protect our codebase from build breaks?

- Git: Improve code quality with branch policies with an option to require that code builds before it can be merged to a branch. This option is not available for GitHub repos.

- TFVC: Use gated check-in.

How do I modify other parts of my build pipeline?

- Specify your build tasks to run tests, scripts, and a wide range of other processes.
- Specify build options such as specifying how completed builds are named, building multiple configurations, creating work items on failure.
- Specify the repository to pick the source of the build and modify options such as how the agent workspace is cleaned.
- Set build triggers to modify how your CI builds run and to specify scheduled builds.
- Specify build retention policies to automatically delete old builds.

I selected parallel multi-configuration, but only one build is running at a time.

If you're using VSTS, you might need more concurrent jobs. See Concurrent build and release jobs in VSTS.

How do I see what has changed in my build pipeline?

View the change history of your build pipeline

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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: App Center Distribute

7/9/2018 • 2 minutes to read • Edit Online

Distribute app builds to testers and users via App Center

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

YAML snippet

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

Arguments

ARGUMENT	DESCRIPTION
App Center connection	(Required) Select the service connection for your Visual Studio App Center connection. To create one, click the Manage link and create a new service connection.
App slug	(Required) The app slug is in the format of **{username}/{app_identifier}**. To locate **{username}** and **{app_identifier}** for an app, click on its name from https://appcenter.ms/apps, and the resulting URL is in the format of https://appcenter.ms/users/**{username}**/apps/**{app_identifier}**. If you are using orgs, the app slug is of the format **{orgname}/{app_identifier}**.
Binary file path	(Required) Relative path from the repo root to the APK or IPA file you want to publish
Symbols type	(Optional) undefined

ARGUMENT	DESCRIPTION
Symbols path	(Optional) Relative path from the repo root to the symbols folder.
Symbols path (*.pdb)	(Optional) Relative path from the repo root to PDB symbols files. Path may contain wildcards.
dSYM path	(Optional) Relative path from the repo root to dSYM folder. Path may contain wildcards.
Mapping file	(Optional) Relative path from the repo root to Android's mapping.txt file.
Include all items in parent folder	(Optional) Upload the selected symbols file or folder and all other items inside the same parent folder. This is required for React Native apps.
Create release notes	(Required) undefined
Release notes	(Required) Release notes for this version.
Release notes file	(Required) Select a UTF-8 encoded text file which contains the Release Notes for this version.
Destination ID	(Optional) ID of the distribution group or store the app will deploy to. Leave it empty to use the default group.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure App Service Deploy

7/17/2018 • 6 minutes to read • Edit Online

Update Azure WebApp Services On Windows, Web App On Linux with built-in images or docker containers, ASP.NET, .NET Core, PHP, Python or Node based Web applications, Function Apps, Mobile Apps, Api applications, Web Jobs using Web Deploy / Kudu REST APIs

YAML snippet

```
# Azure App Service Deploy
# Update Azure WebApp Services On Windows, Web App On Linux with built-in images or docker containers, ASP.NET,
.NET Core, PHP, Python or Node based Web applications, Function Apps, Mobile Apps, Api applications, Web Jobs
using Web Deploy / Kudu REST APIs
- task: AzureRmWebAppDeployment@4
  inputs:
    #connectionType: 'AzureRM' # Options: azureRM, publishProfile
    #azureSubscription: # Required when connectionType == AzureRM
    #publishProfilePath: '$(System.DefaultWorkingDirectory)/**/*.pubxml' # Required when connectionType ==
PublishProfile
    #publishProfilePassword: # Required when connectionType == PublishProfile
    #appType: 'webApp' # Required when connectionType == AzureRM# Options: webApp, webAppLinux,
webAppContainer, functionApp, apiApp, mobileApp
    #webAppName: # Required when connectionType == AzureRM
    #deployToSlotOrASE: false # Optional
    #resourceGroupName: # Required when deployToSlotOrASE == True
    #slotName: 'production' # Required when deployToSlotOrASE == True
    #dockerNamespace: # Required when appType == WebAppContainer
    #dockerRepository: # Required when appType == WebAppContainer
    #dockerImageTag: # Optional
    #virtualApplication: # Optional
    #packageForLinux: '$(System.DefaultWorkingDirectory)/**/*.zip' # Required when connectionType ==
PublishProfile || WebAppKind == WebApp || WebAppKind == ApiApp || WebAppKind == FunctionApp || WebAppKind ==
MobileApp || WebAppKind == WebAppLinux
    #runtimeStack: # Required when appType == WebAppLinux
    #startupCommand: # Optional
    #scriptType: # Optional. Options: , inline Script, file Path
    #inlineScript: ':: You can provide your deployment commands here. One command per line.' # Required when
scriptType == Inline Script
    #scriptPath: # Required when scriptType == File Path
    #webConfigParameters: # Optional
    #appSettings: # Optional
    #configurationSettings: # Optional
    #useWebDeploy: false # Optional
    #deploymentType: 'webDeploy' # Required when useWebDeploy == True# Options: webDeploy, zipDeploy,
runFromZip
    #takeAppOfflineFlag: true # Optional
    #setParametersFile: # Optional
    #removeAdditionalFilesFlag: false # Optional
    #excludeFilesFromAppDataFlag: true # Optional
    #additionalArguments: '-retryAttempts:6 -retryInterval:10000' # Optional
    #renameFilesFlag: true # Optional
    #enableXmlTransform: # Optional
    #enableXmlVariableSubstitution: # Optional
    #jSONFiles: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Connection type	(Required) Select the connection type to use to deploy the WebApp.
Azure subscription	(Required) Select the Azure Resource Manager subscription for the deployment.
Publish profile path	(Required)
Publish profile password	(Required) It is recommended to store password in a secret variable and use that variable here e.g. $(Password).
App Service type	(Required) Choose from Web App On Windows, Web App On Linux, Web App for Containers, Function App and Mobile App.
App Service name	(Required) Enter or Select the name of an existing Azure App Service. App services based on selected app type will only be listed.
Deploy to Slot or App Service Environment	(Optional) Select the option to deploy to an existing deployment slot or Azure App Service Environment. For both the targets, the task needs Resource group name. In case the deployment target is a slot, by default the deployment is done to the production slot. Any other existing slot name can also be provided. In case the deployment target is an Azure App Service environment, leave the slot name as 'production' and just specify the Resource group name.
Resource group	(Required) The Resource group name is required when the deployment target is either a deployment slot or an App Service Environment. Enter or Select the Azure Resource group that contains the Azure App Service specified above.
Slot	(Required) Enter or Select an existing Slot other than the Production slot.
Registry or Namespace	(Required) A globally unique top-level domain name for your specific registry or namespace. Note: Fully qualified image name will be of the format: '`<registry or namespace>`/`<repository>`:`<tag>`'. For example, '**myregistry.azurecr.io**/nginx:latest'.
Image	(Required) Name of the repository where the container images are stored. Note: Fully qualified image name will be of the format: '`<registry or namespace>`/`<repository>`:`<tag>`'. For example, 'myregistry.azurecr.io/**nginx**:latest'.
Tag	(Optional) Tags are optional, it is the mechanism that registries use to give Docker images a version. Note: Fully qualified image name will be of the format: '`<registry or namespace>`/`<repository>`:`<tag>`'. For example, 'myregistry.azurecr.io/nginx:**latest**'.

ARGUMENT	DESCRIPTION
Virtual application	(Optional) Specify the name of the Virtual application that has been configured in the Azure portal. The option is not required for deployments to the App Service root.
Package or folder	(Required) File path to the package or a folder containing app service contents generated by MSBuild or a compressed zip or war file. Variables (Build | Release), wild cards are supported. For example, $(System.DefaultWorkingDirectory)/**/*.zip or $(System.DefaultWorkingDirectory)/**/*.war.
Runtime Stack	(Required) Select the framework and version.
Startup command	(Optional) Enter the start up command.
Deployment script type	(Optional) Customize the deployment by providing a script that will run on the Azure App service once the task has completed the deployment successfully . For example restore packages for Node, PHP, Python applications. Learn more.
Inline Script	(Required) undefined
Deployment script path	(Required) undefined
Generate web.config parameters for Python, Node.js and Go apps	(Optional) A standard Web.config will be generated and deployed to Azure App Service if the application does not have one. The values in web.config can be edited and vary based on the application framework. For example for node.js application, web.config will have startup file and iis_node module values. This edit feature is only for the generated web.config. Learn more.
App settings	(Optional) Edit web app application settings following the syntax -key value . Value containing spaces should be enclosed in double quotes. **Example** : -Port 5000 -RequestTimeout 5000 -WEBSITE_TIME_ZONE "Eastern Standard Time"
Configuration settings	(Optional) Edit web app configuration settings following the syntax -key value. Value containing spaces should be enclosed in double quotes. Example : -phpVersion 5.6 -linuxFxVersion: node|6.11
Take App Offline	(Optional) Select the option to take the Azure App Service offline by placing an app_offline.htm file in the root directory of the App Service before the sync operation begins. The file will be removed after the sync operation completes successfully.
Publish using Web Deploy	(Optional) Publish using Web Deploy options are supported only when using Windows agent. On other platforms, the task relies on Kudu REST APIs to deploy the Azure App Service, and following options are not supported
SetParameters file	(Optional) Optional: location of the SetParameters.xml file to use.

ARGUMENT	DESCRIPTION
Remove additional files at destination	(Optional) Select the option to delete files on the Azure App Service that have no matching files in the App Service package or folder. Note: This will also remove all files related to any extension installed on this Azure App Service. To prevent this, select 'Exclude files from App_Data folder' checkbox.
Exclude files from the App_Data folder	(Optional) Select the option to prevent files in the App_Data folder from being deployed to the Azure App Service.
Additional arguments	(Optional) Additional Web Deploy arguments following the syntax -key:value . These will be applied when deploying the Azure App Service. Example: -disableLink:AppPoolExtension -disableLink:ContentExtension. For more examples of Web Deploy operation settings, refer to this.
Rename locked files	(Optional) Select the option to enable msdeploy flag MSDEPLOY_RENAME_LOCKED_FILES=1 in Azure App Service application settings. The option if set enables msdeploy to rename locked files that are locked during app deployment
XML transformation	(Optional) The config transforms will be run for `*.Release.config` and `*.<EnvironmentName>.config` on the `*.config file`. Config transforms will be run prior to the Variable Substitution. XML transformations are supported only for Windows platform.
XML variable substitution	(Optional) Variables defined in the Build or Release Pipeline will be matched against the 'key' or 'name' entries in the appSettings, applicationSettings, and connectionStrings sections of any config file and parameters.xml. Variable Substitution is run after config transforms. Note: If same variables are defined in the Release Pipeline and in the Environment, then the Environment variables will supersede the Release Pipeline variables.

ARGUMENT	DESCRIPTION

JSON variable substitution	(Optional) Provide new line separated list of JSON files to substitute the variable values. Files names are to be provided relative to the root folder.
	To substitute JSON variables that are nested or hierarchical, specify them using JSONPath expressions.
	
	For example, to replace the value of 'ConnectionString' in the sample below, you need to define a variable as 'Data.DefaultConnection.ConnectionString' in the build/release pipeline (or release pipeline's environment).
	{
	"Data": {
	"DefaultConnection": {
	"ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True"
	}
	}
	}
	Variable Substitution is run after configuration transforms.
	
	Note: Build/release pipeline variables are excluded in substitution

CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure App Service Manage

7/9/2018 • 2 minutes to read • Edit Online

Start, Stop, Restart, Slot swap, Install site extensions or Enable Continuous Monitoring for an Azure App Service

YAML snippet

```
# Azure App Service Manage
# Start, Stop, Restart, Slot swap, Install site extensions or Enable Continuous Monitoring for an Azure App
Service
- task: AzureAppServiceManage@0
  inputs:
    azureSubscription:
    #action: 'Swap Slots' # Optional. Options: swap Slots, start Azure App Service, stop Azure App Service,
restart Azure App Service, install Extensions, enable Continuous Monitoring, start All Continuous Webjobs,
stop All Continuous Webjobs
    webAppName:
    #specifySlotOrASE: false # Optional
    #resourceGroupName: # Required when action == Swap Slots || SpecifySlot == True
    #sourceSlot: # Required when action == Swap Slots
    #swapWithProduction: true # Optional
    #targetSlot: # Required when action == Swap Slots && SwapWithProduction == False
    #preserveVnet: false # Optional
    #slot: 'production' # Required when action != Swap Slots && SpecifySlot == True
    #extensionsList: # Required when action == Install Extensions
    #outputVariable: # Optional
    #appInsightsResourceGroupName: # Required when action == Enable Continuous Monitoring
    #applicationInsightsResourceName: # Required when action == Enable Continuous Monitoring
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription
Action	(Optional) Action to be performed on the App Service. You can Start, Stop, Restart, Slot swap, Install site extensions or enable Continuous Monitoring for an Azure App Service
App Service name	(Required) Enter or select the name of an existing Azure App Service
Specify Slot or App Service Environment	(Optional) undefined
Resource group	(Required) Enter or Select the Azure Resource Group that contains the Azure App Service specified above
Source Slot	(Required) The swap action directs destination slot's traffic to the source slot

ARGUMENT	DESCRIPTION
Swap with Production	(Optional) Select the option to swap the traffic of source slot with production. If this option is not selected, then you will have to provide source and target slot names.
Target Slot	(Required) The swap action directs destination slot's traffic to the source slot
Preserve Vnet	(Optional) Preserve the Virtual network settings
Slot	(Required) undefined
Install Extensions	(Required) Site Extensions run on Microsoft Azure App Service. You can install set of tools as site extension and better manage your Azure App Service. The App Service will be restarted to make sure latest changes take effect.
Output variable	(Optional) Provide the variable name for the local installation path for the selected extension. This field is now deprecated and would be removed. Use LocalPathsForInstalledExtensions variable from Output Variables section in subsequent tasks.
Resource Group name for Application Insights	(Required) Enter or Select resource group where your application insights resource is available
Application Insights resource name	(Required) Select Application Insights resource where continuous monitoring data will be recorded. If your application insights resource is not listed here and you want to create a new resource, click on [+New] button. Once the resource is created on Azure Portal, come back here and click on refresh button.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure Cloud Service Deployment

7/9/2018 • 2 minutes to read • Edit Online

Deploy an Azure Cloud Service

YAML snippet

```
# Azure Cloud Service Deployment
# Deploy an Azure Cloud Service
- task: AzureCloudPowerShellDeployment@1
  inputs:
    azureClassicSubscription:
    storageAccount:
    serviceName:
    serviceLocation:
    csPkg:
    csCfg:
    #slotName: 'Production'
    #deploymentLabel: '$(Build.BuildNumber)' # Optional
    #appendDateTimeToLabel: false # Optional
    #allowUpgrade: true
    #simultaneousUpgrade: false # Optional
    #forceUpgrade: false # Optional
    #diagnosticStorageAccountKeys: # Optional
    #newServiceCustomCertificates: # Optional
    #newServiceAdditionalArguments: # Optional
    #newServiceAffinityGroup: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription (Classic)	(Required) Azure Classic subscription to target for deployment.
Storage account	(Required) Storage account must exist prior to deployment.
Service name	(Required) Select or enter an existing cloud service name.
Service location	(Required) Select a region for new service deployment.Possible options are **East US**, **East US 2**, **Central US**, **South Central US**, **West US**, **North Europe**, **West Europe** and others.
CsPkg	(Required) Path of CsPkg under the default artifact directory.
CsCfg	(Required) Path of CsCfg under the default artifact directory.
Environment (Slot)	(Required) **Production** or **Staging**
Deployment label	(Optional) Specifies the label name for the new deployment. If not specified, a Globally Unique Identifier (GUID) is used.

ARGUMENT	DESCRIPTION
Append current date and time	(Optional) Appends current date and time to deployment label
Allow upgrade	(Required) When selected allows an upgrade to the Microsoft Azure deployment
Simultaneous upgrade	(Optional) Updates all instances at once. Your cloud service will be unavailable during update.
Force upgrade	(Optional) When selected sets the upgrade to a forced upgrade, which could potentially cause loss of local data.
Diagnostic storage account keys	(Optional) Provide storage keys for diagnostics storage account in Role:Storagekey format. The diagnostics storage account name for each role will be obtained from diagnostics config file (.wadcfgx). If the .wadcfgx file for a role is not found, diagnostics extensions won't be set for the role. If the storage account name is missing in the .wadcfgx file, the default storage account will be used for storing diagnostics results and the storage key parameters from deployment task will be ignored. It's recommended to save <storage_account_key> as a secret variable unless there is no sensitive information in the diagnostics result for your environment. For example, WebRole: <WebRole_storage_account_key> WorkerRole: <WorkerRole_stoarge_account_key>
Custom certificates to import	(Optional) Provide custom certificates in CertificatePfxBase64:CertificatePassword format. It's recommended to save <certificate_password> as a secret variable. For example, Certificate1: <Certificate1_password> Certificate2: <Certificate2_password>
Additional arguments	(Optional) Pass in additional arguments while creating a brand new service. These will be passed on to `New-AzureService` cmdlet. Eg: `-Label 'MyTestService'`
Affinity group	(Optional) While creating new service, this affinity group will be considered instead of using service location.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure CLI

7/9/2018 • 3 minutes to read • Edit Online

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, Microsoft-hosted agents do not have AZ CLI installed. You can either install using `npm install -g azure-cli` or use self-hosted agents with AZ CLI pre-installed

Demands

None

Prerequisites

- A Microsoft Azure subscription
- A service connection to your Azure account. You can use either:
  - Azure Classic service connection
  - Azure Resource Manager service connection
- Azure CLI installed on the computer(s) that run the build and release agent. See Install the Azure CLI. If an agent is already running on the machine on which the Azure CLI is installed, restart the agent to ensure all the relevent environment variables are updated.

YAML snippet

```
# Azure CLI
# Run a Shell or Batch script with Azure CLI commands against an azure subscription
- task: AzureCLI@1
  inputs:
    azureSubscription:
    #scriptLocation: 'scriptPath' # Options: inlineScript, scriptPath
    #scriptPath: # Required when scriptLocation == ScriptPath
    #inlineScript: # Required when scriptLocation == InlineScript
    #arguments: # Optional
    #workingDirectory: # Optional
    #failOnStandardError: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Connection Type	Required. Select the type of service connection used to define the connection to Azure. Choose **Azure Classic** or **Azure Resource Manager**. This parameter is shown only when the selected task version is 0.* as Azure CLI task v1.0 supports only Azure Resource Manager (ARM) subscriptions.
Azure Classic Subscription	Required if you select **Azure Classic** for the **Azure Connection Type** parameter. The name of an Azure Classic service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located.
Azure RM Subscription	Required if you select **Azure Resource Manager** for the **Azure Connection Type** parameter. The name of an Azure Resource Manager service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located. See Azure Resource Manager overview for more details.
Script Location	Required. The way that the script is provided. Choose **Inline Script** or **Script Path** (the default).
Inline Script	Required if you select **Inline Script** for the **Script Location** parameter. Type or copy the script code to execute here. You can include default variables, global variables, and environment variables.
Script Path	Required if you select **Script Path** for the **Script Location** parameter. The path to a linked artifact that is the **.bat**, **.cmd**, or **.sh** script you want to run. It can be a fully-qualified path, or a valid path relative to the default working directory.
Arguments	Optional. Any arguments you want to pass to the script.
Advanced - Working Directory	Optional. The working directory in which the script will execute. If not specified, this will be the folder containing the script file.
Advanced - Fail on Standard Error	Set this option if you want the build to fail if errors are written to the **StandardError** stream.

ARGUMENT	DESCRIPTION
Control options	See Control options

Related tasks

- Azure Resource Group Deployment
- Azure Cloud Service Deployment
- Azure Web App Deployment

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

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 File Copy

7/10/2018 • 10 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015.3

Copy files to Microsoft Azure storage blobs or virtual machines (VMs).

> **NOTE**
>
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

The task is used to copy application files and other artifacts that are required in order to install the app; such as PowerShell scripts, PowerShell-DSC modules, and more.

When the target is Azure VMs, the files are first copied to an automatically generated Azure blob container and then downloaded into the VMs. The container is deleted after the files have been successfully copied to the VMs.

The task uses **AzCopy**, the command-line utility built for fast copying of data from and into Azure storage accounts.

To dynamically deploy Azure Resource Groups that contain virtual machines, use the Azure Resource Group Deployment task. This task has a sample template that can perform the required operations to set up the WinRM HTTPS protocol on the virtual machines, open the 5986 port in the firewall, and install the test certificate.

Demands

None

YAML snippet

```
# Azure File Copy
# Copy files to Azure blob or VM(s)
- task: AzureFileCopy@2
  inputs:
    sourcePath:
    #azureConnectionType: 'ConnectedServiceNameARM' # Optional. Options: connectedServiceName,
connectedServiceNameARM
    #azureClassicSubscription: # Required when azureConnectionType == ConnectedServiceName
    #azureSubscription: # Required when azureConnectionType == ConnectedServiceNameARM
    destination: # Options: azureBlob, azureVMs
    #classicStorage: # Required when azureConnectionType == ConnectedServiceName
    #storage: # Required when azureConnectionType == ConnectedServiceNameARM
    #containerName: # Required when destination == AzureBlob
    #blobPrefix: # Optional
    #cloudService: # Required when azureConnectionType == ConnectedServiceName && Destination == AzureVMs
    #resourceGroup: # Required when azureConnectionType == ConnectedServiceNameARM && Destination == AzureVMs
    #resourceFilteringMethod: 'machineNames' # Optional. Options: machineNames, tags
    #machineNames: # Optional
    #vmsAdminUserName: # Required when destination == AzureVMs
    #vmsAdminPassword: # Required when destination == AzureVMs
    #targetPath: # Required when destination == AzureVMs
    #additionalArgumentsForBlobCopy: # Optional
    #additionalArgumentsForVMCopy: # Optional
    #enableCopyPrerequisites: false # Optional
    #copyFilesInParallel: true # Optional
    #cleanTargetBeforeCopy: false # Optional
    #skipCACheck: true # Optional
    #outputStorageUri: # Optional
    #outputStorageContainerSasToken: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Source	Required. The source of the files to copy. Pre-defined system variables such as `$(Build.Repository.LocalPath)` can be used. Names containing wildcards such as `*.zip` are not supported.
Azure Connection Type	Required. Select the type of service connection used to define the connection to Azure. Choose **Azure Classic** or **Azure Resource Manager**.
Azure Classic Subscription	Required if you select **Azure Classic** for the **Azure Connection Type** parameter. The name of an Azure Classic service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located.
Azure RM Subscription	Required if you select **Azure Resource Manager** for the **Azure Connection Type** parameter. The name of an Azure Resource Manager service connection configured for the subscription where the target Azure service, virtual machine, or storage account is located. See Azure Resource Manager overview for more details.
Destination Type	Required. The type of target destination for the files. Choose **Azure Blob** or **Azure VMs**.

ARGUMENT	DESCRIPTION
Classic Storage Account	Required if you select **Azure Classic** for the **Azure Connection Type** parameter. The name of an existing storage account within the Azure subscription.
RM Storage Account	Required if you select **Azure Resource Manager** for the **Azure Connection Type** parameter. The name of an existing storage account within the Azure subscription.
Container Name	Required if you select **Azure Blob** for the **Destination Type** parameter. The name of the container to which the files will be copied. If a container with this name does not exist, a new one will be created.
Blob Prefix	Optional if you select **Azure Blob** for the **Destination Type** parameter. A prefix for the blob names, which can be used to filter the blobs. For example, using the build number enables easy filtering when downloading all blobs with the same build number.
Cloud Service	Required if you select **Azure Classic** for the **Azure Connection Type** parameter and **Azure VMs** for the **Destination Type** parameter. The name of the Azure cloud service in which the virtual machines run.
Resource Group	Required if you select **Azure Resource Manager** for the **Azure Connection Type** parameter and **Azure VMs** for the **Destination Type** parameter. The name of the Azure Resource Group in which the virtual machines run.
Select Machines By	Depending on how you want to specify the machines in the group when using the **Filter Criteria** parameter, choose **Machine Names** or **Tags**.
Filter Criteria	Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be: - The name of an Azure Resource Group. - An output variable from a previous task. - A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses. Specify tag names for a filter as {TagName}**:**{Value} Example: `Role:DB;OS:Win8.1`
Admin Login	Required if you select **Azure VMs** for the **Destination Type** parameter. The user name of an account that has administrative permissions for all the target VMs. - Formats such as **username**, **domain\username**, **machine-name\username**, and **.\username** are supported. - UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.
Password	Required if you select **Azure VMs** for the **Destination Type** parameter. The password for the account specified as the **Admin Login** parameter. Use the padlock icon for a variable defined in the **Variables** tab to protect the value, and insert the variable name here.

ARGUMENT	DESCRIPTION
Destination Folder	Required if you select **Azure VMs** for the **Destination Type** parameter. The folder in the Azure VMs to which the files will be copied. Environment variables such as `$env:windir` and `$env:systemroot` are supported. Examples: `$env:windir\FabrikamFiber\Web` and `c:\FabrikamFiber`
Additional Arguments	Optional. Any arguments you want to pass to the **AzCopy.exe** program for use when uploading to the blob and downloading to the VMs. See Transfer data with the AzCopy Command-Line Utility for more details. If you are using a Premium storage account, which supports only Azure page blobs, the pass `/BlobType:Page` as an additional argument.
Enable Copy Prerequisites	Available if you select **Azure Resource Manager** for the **Azure Connection Type** parameter and **Azure VMs** for the **Destination Type** parameter. Setting this option configures the Windows Remote Management (WinRM) listener over HTTPS protocol on port 5986, using a self-signed certificate. This configuration is required for performing copy operation on Azure virtual machines. - If the target virtual machines are accessed through a load balancer, ensure an inbound NAT rule is configured to allow access on port 5986. - If the target virtual machines are associated with a Network Security Group (NSG), configure an inbound security rule to allow access on port 5986.
Copy in Parallel	Available if you select **Azure VMs** for the **Destination Type** parameter. Setting this option causes the process to execute in parallel for the copied files. This can considerably reduce the overall time taken.
Clean Target	Available if you select **Azure VMs** for the **Destination Type** parameter. Setting this option causes all of the files in the destination folder to be deleted before the copy process starts.
Test Certificate	Available if you select **Azure VMs** for the **Destination Type** parameter. WinRM requires a certificate for the HTTPS transfer when copying files from the intermediate storage blob into the Azure VMs. If you set use a self-signed certificate, set this option to prevent the process from validating the certificate with a trusted certificate authority (CA).
Output - Storage Container URI	Optional. The name of a variable that will be updated with the URI of the storage container into which the files were copied. Use this variable as an input to subsequent tasks.
Output - Storage Container SAS Token	Optional. The name of a variable that will be updated with the Storage Access Security (SAS) token of the storage container into which the files were copied. Use this variable as an input to subsequent tasks. By default, the SAS token expires after 4 hours.
Control options	See Control options

Related tasks

- Azure Resource Group Deployment
- Azure Cloud Service Deployment
- Azure Web App Deployment

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

What are the Azure PowerShell prerequisites for using this task?

The task requires Azure PowerShell to be installed on the machine running the automation agent. The recommended version is 1.0.2, but the task will work with version 0.9.8 and higher. You can use the Azure PowerShell Installer v1.0.2 to obtain this.

What are the WinRM prerequisites for this task?

The task uses Windows Remote Management (WinRM) HTTPS protocol to copy the files from the storage blob container to the Azure VMs. This requires the WinRM HTTPS service to be configured on the VMs, and a suitable certificate installed.

If the VMs have been created without opening the WinRM HTTPS ports, follow these steps:

1. Configure an inbound access rule to allow HTTPS on port 5986 of each VM.
2. Disable UAC remote restrictions.
3. Specify the credentials for the task to access the VMs using an administrator-level login in the simple form **username** without any domain part.
4. Install a certificate on the machine that runs the automation agent.
5. Set the **Test Certificate** parameter of the task if you are using a self-signed certificate.

For more details, see this blog post.

What type of service connection should I choose?

The following table lists the storage accounts and the service connections that work with them. To identify whether a storage account is based on the classic APIs or the Resource Manager APIs, log into the Azure portal and browse for **Storage accounts (Classic)** or **Storage accounts**.

STORAGE ACCOUNT TYPE	AZURE SERVICE CONNECTIONS IN TFS/TS
Resource Manager	Azure Resource Manager service connection
Classic	Azure service connection with certificate-based or credentials-based authentication using a school or work account

- For Azure classic resources, use an **Azure** service connection type with certificate or credentials-based authentication. If you are using credentials-based authentication, ensure that the credentials are for a school or work account. Microsoft accounts such as **joe@live.com** and **joe@hotmail.com** are not supported.
- For Azure Resource Manager VMs, use an **Azure Resource Manager** service connection type. See more details at Automating Azure Resource Group deployment using a Service Principal.
- If you are using an **Azure Resource Manager** service connection type, or an **Azure** service connection type with certificate-based authentication, the task automatically filters appropriate classic storage accounts, the newer Azure Resource Manager storage accounts, and other fields. For example, the Resource Group or cloud service, and the virtual machines.

- **Note**: Currently an **Azure** service connection type with credentials-based authentication does not filter the storage, Resource Group or cloud service, and virtual machine fields.

What happens if my Resource Group contains both Classic and Resource Manager VMs?

If the specified Resource Group contains both Azure Resource Manager and Classic VMs, the set of VMs that will be targeted depends on the connection type. For certificate-based connections and credentials-based connections, the copy operation will be performed only on Classic VMs. For Service Principal Name based connections, the copy operation will be performed on only Resource Manager VMs.

How do I create a school or work account for use with this task?

A suitable account can be easily created for use in a service connection:

1. Use the Azure portal to create a new user account in Azure Active Directory.
2. Add the Azure Active Directory user account to the co-administrators group in your Azure subscription.
3. Sign into the Azure portal with this user account and change the password.
4. Use the username and password of this account in the service connection. Deployments will be processed using this account.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/10/2018 • 3 minutes 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 pipeline. The task is Node-based, and works with agents on Linux, macOS, and Windows.

Prerequisites

The task has the following pre-requisites:

- An Azure subscription linked to Team Foundation Server or VSTS using the Azure Resource Manager service connection.
- An Azure Key Vault containing the secrets.

You can create a key vault:

- In the Azure portal
- By using Azure PowerShell
- By using the Azure CLI

Add secrets to a key vault:

- By using the PowerShell cmdlet Set-AzureKeyVaultSecret. If the secret does not exist, this cmdlet creates it. If the secret already exists, this cmdlet creates a new version of that secret.
- By using the Azure CLI. To add a secret to a key vault, for example a secret named **SQLPassword** with the value **Pa$$w0rd**, type:

```
az keyvault secret set --vault-name 'ContosoKeyVault' --name 'SQLPassword' --value 'Pa$$w0rd'
```

When you want to access secrets:

- Ensure the Azure service connection has at least **Get** and **List** permissions on the vault. You can set these permissions in the Azure portal:
  - Open the **Settings** blade for the vault, choose **Access policies**, then **Add new**.
  - In the **Add access policy** blade, choose **Select principal** and select the service principal for your client account.
  - In the **Add access policy** blade, choose **Secret permissions** and ensure that **Get** and **List** are checked (ticked).
  - Choose **OK** to save the changes.

YAML snippet

```
# Azure Key Vault
# Download Azure Key Vault Secrets
- task: AzureKeyVault@1
  inputs:
    azureSubscription:
    keyVaultName:
    #secretsFilter: '*' # Options: editableOptions
```

Arguments

PARAMETER	DESCRIPTION
Azure Subscription	Required. Select the service connection for the Azure subscription containing the Azure Key Vault instance, or create a new connection. Learn more
Key Vault	Required. Select the name of the Azure Key Vault from which the secrets will be downloaded.
Secrets filter	Required. A comma-separated list of secret names to be downloaded. Use the default value `*` 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_Queries@microsoft.com if you discover issues using the task, to share feedback about the task, or to suggest new features that you would like to see.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

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 Monitor Alerts

7/9/2018 • 2 minutes to read • Edit Online

Configure alerts on available metrics for an Azure resource

YAML snippet

```
# Azure Monitor Alerts
# Configure alerts on available metrics for an Azure resource
- task: AzureMonitorAlerts@0
  inputs:
    azureSubscription:
    resourceGroupName:
    #resourceType: 'Microsoft.Insights/components' # Options: microsoft.Insights/Components,
microsoft.Web/Sites, microsoft.Storage/StorageAccounts, microsoft.Compute/VirtualMachines
    resourceName:
    alertRules:
    #notifyServiceOwners: # Optional
    #notifyEmails: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Subscription	(Required) Select the Azure Resource Manager subscription. Note: To configure new service connection, select the Azure subscription from the list and click 'Authorize'. If your subscription is not listed or if you want to use an existing Service Principal, you can setup an Azure service connection using 'Add' or 'Manage' button.
Resource Group	(Required) Select the Azure Resource Group that contains the Azure resource where you want to configure an alert.
Resource Type	(Required) Select the Azure resource type.
Resource name	(Required) Select name of Azure resource where you want to configure an alert.
Alert rules	(Required) List of Azure monitor alerts configured on selected Azure resource. To add or modify alerts, click on [...] button.
Subscription owners, contributors and readers	(Optional) Send email notification to everyone who has access to this resource group.
Additional administrator emails	(Optional) Add additional email addresses separated by semicolons(;) if you want to send email notification to additional people (whether or not you checked the "subscription owners..." box).
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure Database for Mysql Deployment

7/9/2018 • 2 minutes to read • Edit Online

This is an early preview. Run your scripts and make changes to your Azure DB for Mysql.

YAML snippet

```
# Azure Database for Mysql Deployment
# This is an early preview. Run your scripts and make changes to your Azure DB for Mysql.
- task: AzureMysqlDeployment@1
  inputs:
    azureSubscription:
    serverName:
    #databaseName: # Optional
    sqlUsername:
    sqlPassword:
    #taskNameSelector: 'SqlTaskFile' # Optional. Options: sqlTaskFile, inlineSqlTask
    #sqlFile: # Required when taskNameSelector == SqlTaskFile
    #sqlInline: # Required when taskNameSelector == InlineSqlTask
    #sqlAdditionalArguments: # Optional
    #ipDetectionMethod: 'AutoDetect' # Options: autoDetect, iPAddressRange
    #startIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #endIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #deleteFirewallRule: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Subscription	(Required) This is needed to connect to your Azure account. To configure new service connection, select the Azure subscription from the list and click 'Authorize'. If your subscription is not listed or if you want to use an existing Service Principal, you can setup an Azure service connection using 'Add' or 'Manage' button.
Host Name	(Required) Server name of 'Azure DB for Mysql'.Example: fabrikam.mysql.database.azure.com. When you connect using Mysql Workbench, this is the same value that is used for 'Hostname' in 'Parameters'
Database Name	(Optional) The name of database, if you already have one, on which the below script is needed to be run, else the script itself can be used to create the database.
Server Admin Login	(Required) Azure Database for MySQL server supports native MySQL authentication. You can connect and authenticate to a server with the server's admin login. Example: bbo1@fabrikam. When you connect using Mysql Workbench, this is the same value that is used for 'Username' in 'Parameters'.

ARGUMENT	DESCRIPTION
Password	(Required) Administrator password for Azure DB for Mysql. In case you don't recall the password you can change the password from Azure portal. It can be variable defined in the pipeline. Example : $(password). Also, you may mark the variable type as 'secret' to secure it.
Type	(Optional) Select one of the options between Script File & Inline Script.
MySQL Script	(Required) Full path of the script file on the automation agent or on a UNC path accessible to the automation agent like, \\BudgetIT\DeployBuilds\script.sql. Also, predefined system variables like, $(agent.releaseDirectory) can also be used here. A file containing SQL statements can be used here.
Inline MySQL Script	(Required) Enter the MySQL script to execute on the Database selected above.
Additional Mysql Arguments	(Optional) Additional options supported by mysql simple SQL shell. These options will be applied when executing the given file on the Azure DB for Mysql. Example: You can change to default tab separated output format to HTML or even XML format. Or if you have problems due to insufficient memory for large result sets, use the --quick option.
Specify Firewall Rules Using	(Required) For successful execution of the task, we need to enable administrators to access the Azure Database for MySQL Server from the IP Address of the automation agent. By selecting auto-detect you can automatically add firewall exception for range of possible IP Address of automation agent or else you can specify the range explicitly.
Start IP Address	(Required) The starting IP Address of the automation agent machine pool like 196.21.30.50 .
End IP Address	(Required) The ending IP Address of the automation agent machine pool like 196.21.30.65 .
Delete Rule After Task Ends	(Optional) If selected, the added exception for IP addresses of the automation agent will be removed for corresponding Azure Database for MySQL.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure PowerShell

7/9/2018 • 2 minutes to read • Edit Online

Run a PowerShell script within an Azure environment

YAML snippet

```
# Azure PowerShell
# Run a PowerShell script within an Azure environment
- task: AzurePowerShell@3
  inputs:
    #azureConnectionType: 'ConnectedServiceNameARM' # Optional. Options: connectedServiceName,
connectedServiceNameARM
    #azureClassicSubscription: # Required when azureConnectionType == ConnectedServiceName
    #azureSubscription: # Required when azureConnectionType == ConnectedServiceNameARM
    #scriptType: 'FilePath' # Optional. Options: filePath, inlineScript
    #scriptPath: # Optional
    #inline: '# You can write your azure powershell scripts inline here. # You can also pass predefined and
custom variables to this script using arguments' # Optional
    #scriptArguments: # Optional
    #errorActionPreference: 'stop' # Optional. Options: stop, continue, silentlyContinue
    #failOnStandardError: false # Optional
    #azurePowerShellVersion: 'OtherVersion' # Optional. Options: latestVersion, otherVersion
    #preferredAzurePowerShellVersion: # Required when azurePowerShellVersion == OtherVersion
```

Arguments

ARGUMENT	DESCRIPTION
Azure Connection Type	(Optional)
Azure Classic Subscription	(Required) Azure Classic subscription to configure before running PowerShell
Azure Subscription	(Required) Azure Resource Manager subscription to configure before running PowerShell
Script Type	(Optional) Type of the script: File Path or Inline Script
Script Path	(Optional) Path of the script. Should be fully qualified path or relative to the default working directory.
Inline Script	(Optional) Enter the script to execute.
Script Arguments	(Optional) Additional parameters to pass to PowerShell. Can be either ordinal or named parameters.
ErrorActionPreference	(Optional) Select the value of the ErrorActionPreference variable for executing the script.

ARGUMENT	DESCRIPTION
Fail on Standard Error	(Optional) If this is true, this task will fail if any errors are written to the error pipeline, or if any data is written to the Standard Error stream.
Azure PowerShell Version	(Optional) In case of Microsoft-hosted agents, the supported Azure PowerShell Versions are: 2.1.0, 3.8.0, 4.2.1 and 5.1.1(Hosted VS2017 Queue), 3.6.0(Hosted Queue). To pick the latest version available on the agent, select "Latest installed version". For self-hosted agents you can specify preferred version of Azure PowerShell using "Specify version"
Preferred Azure PowerShell Version	(Required) Preferred Azure PowerShell Version needs to be a proper semantic version eg. 1.2.3. Regex like 2.*,2.3.* is not supported. The Hosted VS2017 Pool currently supports versions: 2.1.0, 3.8.0, 4.2.1, 5.1.1
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure Resource Group Deployment

7/17/2018 • 5 minutes to read • Edit Online

Deploy, start, stop, delete Azure Resource Groups

YAML snippet

```
# Azure Resource Group Deployment
# Deploy an Azure resource manager (ARM) template to a resource group. You can also start, stop, delete,
deallocate all Virtual Machines (VM) in a resource group
- task: AzureResourceGroupDeployment@2
  inputs:
    azureSubscription:
    #action: 'Create Or Update Resource Group' # Options: create Or Update Resource Group, select Resource
Group, start, stop, stopWithDeallocate, restart, delete, deleteRG
    resourceGroupName:
    #location: # Required when action == Create Or Update Resource Group
    #templateLocation: 'Linked artifact' # Options: linked Artifact, uRL Of The File
    #csmFileLink: # Required when templateLocation == URL Of The File
    #csmParametersFileLink: # Optional
    #csmFile: # Required when  TemplateLocation == Linked Artifact
    #csmParametersFile: # Optional
    #overrideParameters: # Optional
    #deploymentMode: 'Incremental' # Options: incremental, complete, validation
    #enableDeploymentPrerequisites: 'None' # Optional. Options: none, configureVMwithWinRM,
configureVMWithDGAgent
    #teamServicesConnection: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
    #teamProject: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
    #deploymentGroupName: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent
    #copyAzureVMTags: # Optional
    #runAgentServiceAsUser: # Optional
    #userName: # Required when enableDeploymentPrerequisites == ConfigureVMWithDGAgent &&
RunAgentServiceAsUser == True
    #password: # Optional
    #outputVariable: # Optional
    #deploymentOutputs: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription for the deployment.
Action	(Required) Action to be performed on the Azure resources or resource group.
Resource group	(Required) Provide the name of a resource group.
Location	(Required) Location for deploying the resource group. If the resource group already exists in the subscription, then this value will be ignored.
Template location	(Required) undefined

ARGUMENT	DESCRIPTION
Template link	(Required) Specify the URL of the template file. Example: https://raw.githubusercontent.com/Azure/... To deploy a template stored in a private storage account, retrieve and include the shared access signature (SAS) token in the URL of the template. Example: `<blob_storage_url>/template.json?<SAStoken>` To upload a template file (or a linked template) to a storage account and generate a SAS token, you could use Azure file copy task or follow the steps using PowerShell or Azure CLI. To view the template parameters in a grid, click on "..." next to Override template parameters text box. This feature requires that CORS rules are enabled at the source. If templates are in Azure storage blob, refer to this to enable CORS.
Template parameters link	(Optional) Specify the URL of the parameters file. Example: https://raw.githubusercontent.com/Azure/... To use a file stored in a private storage account, retrieve and include the shared access signature (SAS) token in the URL of the template. Example: `<blob_storage_url>/template.json?<SAStoken>` To upload a parameters file to a storage account and generate a SAS token, you could use Azure file copy task or follow the steps using PowerShell or Azure CLI. To view the template parameters in a grid, click on "..." next to Override template parameters text box. This feature requires that CORS rules are enabled at the source. If templates are in Azure storage blob, refer to this to enable CORS.
Template	(Required) Specify the path or a pattern pointing to the Azure Resource Manager template. For more information about the templates see https://aka.ms/azuretemplates. To get started immediately use template https://aka.ms/sampletemplate.
Template parameters	(Optional) Specify the path or a pattern pointing for the parameters file for the Azure Resource Manager template.
Override template parameters	(Optional) To view the template parameters in a grid, click on "..." next to Override Parameters textbox. This feature requires that CORS rules are enabled at the source. If templates are in Azure storage blob, refer to this to enable CORS. Or type the template parameters to override in the textbox. Example, –storageName fabrikam –adminUsername $(vmusername) -adminPassword $(password) –azureKeyVaultName $(fabrikamFibre). If the parameter value you're using has multiple words, enclose them in quotes, even if you're passing them using variables. For example, -name "parameter value" -name2 "$(var)" To override object type parameters use stringified JSON objects. For example, -options ["option1"] -map {"key1": "value1" }.

ARGUMENT	DESCRIPTION
Deployment mode	(Required) Incremental mode handles deployments as incremental updates to the resource group . It leaves unchanged resources that exist in the resource group but are not specified in the template. Complete mode deletes resources that are not in your template. Validate mode enables you to find problems with the template before creating actual resources. By default, Incremental mode is used.
Enable prerequisites	(Optional) These options would be applicable only when the Resource group contains virtual machines. Choosing Deployment Group option would configure Deployment Group agent on each of the virtual machines. Selecting WinRM option configures Windows Remote Management (WinRM) listener over HTTPS protocol on port 5986, using a self-signed certificate. This configuration is required for performing deployment operation on Azure machines. If the target Virtual Machines are backed by a Load balancer, ensure Inbound NAT rules are configured for target port (5986).
TFS/VSTS endpoint	(Required) Agent registration with a deployment group requires access to your Visual Studio project. Click "Add" to create a service connection using a personal access token (PAT) with scope restricted to "Deployment Group" and a default expiration time of 90 days. Click "Manage" to update service connection details.
Project	(Required) Specify the project which has the Deployment Group defined in it
Deployment Group	(Required) Specify the Deployment Group against which the Agent(s) will be registered. For more guidance, refer to Deployment Groups
Copy Azure VM tags to agents	(Optional) Choose if the tags configured on the Azure VM need to be copied to the corresponding Deployment Group agent. By default all Azure tags will be copied following the format “Key: Value”. Example: An Azure Tag “Role : Web” would be copied as-is to the Agent machine. For more information on how tag Azure resources refer to link
VM details for WinRM	(Optional) Provide a name for the variable for the resource group. The variable can be used as $(variableName) to refer to the resource group in subsequent tasks like in the PowerShell on Target Machines task for deploying applications. Valid only when the selected action is Create, Update or Select, and required when an existing resource group is selected.

ARGUMENT	DESCRIPTION
Deployment outputs	(Optional) Provide a name for the variable for the output variable which will contain the outputs section of the current deployment object in string format. You can use the "ConvertFrom-Json" PowerShell cmdlet to parse the JSON object and access the individual output values.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure App Service Deploy

7/17/2018 • 6 minutes to read • Edit Online

Update Azure WebApp Services On Windows, Web App On Linux with built-in images or docker containers, ASP.NET, .NET Core, PHP, Python or Node based Web applications, Function Apps, Mobile Apps, Api applications, Web Jobs using Web Deploy / Kudu REST APIs

YAML snippet

```
# Azure App Service Deploy
# Update Azure WebApp Services On Windows, Web App On Linux with built-in images or docker containers,
ASP.NET, .NET Core, PHP, Python or Node based Web applications, Function Apps, Mobile Apps, Api applications,
Web Jobs using Web Deploy / Kudu REST APIs
- task: AzureRmWebAppDeployment@4
  inputs:
    #connectionType: 'AzureRM' # Options: azureRM, publishProfile
    #azureSubscription: # Required when connectionType == AzureRM
    #publishProfilePath: '$(System.DefaultWorkingDirectory)/**/*.pubxml' # Required when connectionType ==
PublishProfile
    #publishProfilePassword: # Required when connectionType == PublishProfile
    #appType: 'webApp' # Required when connectionType == AzureRM# Options: webApp, webAppLinux,
webAppContainer, functionApp, apiApp, mobileApp
    #webAppName: # Required when connectionType == AzureRM
    #deployToSlotOrASE: false # Optional
    #resourceGroupName: # Required when deployToSlotOrASE == True
    #slotName: 'production' # Required when deployToSlotOrASE == True
    #dockerNamespace: # Required when appType == WebAppContainer
    #dockerRepository: # Required when appType == WebAppContainer
    #dockerImageTag: # Optional
    #virtualApplication: # Optional
    #packageForLinux: '$(System.DefaultWorkingDirectory)/**/*.zip' # Required when connectionType ==
PublishProfile || WebAppKind == WebApp || WebAppKind == ApiApp || WebAppKind == FunctionApp || WebAppKind ==
MobileApp || WebAppKind == WebAppLinux
    #runtimeStack: # Required when appType == WebAppLinux
    #startupCommand: # Optional
    #scriptType: # Optional. Options: , inline Script, file Path
    #inlineScript: ':: You can provide your deployment commands here. One command per line.' # Required when
scriptType == Inline Script
    #scriptPath: # Required when scriptType == File Path
    #webConfigParameters: # Optional
    #appSettings: # Optional
    #configurationSettings: # Optional
    #useWebDeploy: false # Optional
    #deploymentType: 'webDeploy' # Required when useWebDeploy == True# Options: webDeploy, zipDeploy,
runFromZip
    #takeAppOfflineFlag: true # Optional
    #setParametersFile: # Optional
    #removeAdditionalFilesFlag: false # Optional
    #excludeFilesFromAppDataFlag: true # Optional
    #additionalArguments: '-retryAttempts:6 -retryInterval:10000' # Optional
    #renameFilesFlag: true # Optional
    #enableXmlTransform: # Optional
    #enableXmlVariableSubstitution: # Optional
    #jSONFiles: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Connection type	(Required) Select the connection type to use to deploy the WebApp.
Azure subscription	(Required) Select the Azure Resource Manager subscription for the deployment.
Publish profile path	(Required)
Publish profile password	(Required) It is recommended to store password in a secret variable and use that variable here e.g. $(Password).
App Service type	(Required) Choose from Web App On Windows, Web App On Linux, Web App for Containers, Function App and Mobile App.
App Service name	(Required) Enter or Select the name of an existing Azure App Service. App services based on selected app type will only be listed.
Deploy to Slot or App Service Environment	(Optional) Select the option to deploy to an existing deployment slot or Azure App Service Environment. For both the targets, the task needs Resource group name. In case the deployment target is a slot, by default the deployment is done to the production slot. Any other existing slot name can also be provided. In case the deployment target is an Azure App Service environment, leave the slot name as 'production' and just specify the Resource group name.
Resource group	(Required) The Resource group name is required when the deployment target is either a deployment slot or an App Service Environment. Enter or Select the Azure Resource group that contains the Azure App Service specified above.
Slot	(Required) Enter or Select an existing Slot other than the Production slot.
Registry or Namespace	(Required) A globally unique top-level domain name for your specific registry or namespace. Note: Fully qualified image name will be of the format: '`<registry or namespace>`/`<repository>`:`<tag>`'. For example, '**myregistry.azurecr.io**/nginx:latest'.
Image	(Required) Name of the repository where the container images are stored. Note: Fully qualified image name will be of the format: '`<registry or namespace>`/`<repository>`:`<tag>`'. For example, 'myregistry.azurecr.io/**nginx**:latest'.
Tag	(Optional) Tags are optional, it is the mechanism that registries use to give Docker images a version. Note: Fully qualified image name will be of the format: '`<registry or namespace>`/`<repository>`:`<tag>`'. For example, 'myregistry.azurecr.io/nginx:**latest**'.

ARGUMENT	DESCRIPTION
Virtual application	(Optional) Specify the name of the Virtual application that has been configured in the Azure portal. The option is not required for deployments to the App Service root.
Package or folder	(Required) File path to the package or a folder containing app service contents generated by MSBuild or a compressed zip or war file. Variables (Build | Release), wild cards are supported. For example, $(System.DefaultWorkingDirectory)/**/*.zip or $(System.DefaultWorkingDirectory)/**/*.war.
Runtime Stack	(Required) Select the framework and version.
Startup command	(Optional) Enter the start up command.
Deployment script type	(Optional) Customize the deployment by providing a script that will run on the Azure App service once the task has completed the deployment successfully . For example restore packages for Node, PHP, Python applications. Learn more.
Inline Script	(Required) undefined
Deployment script path	(Required) undefined
Generate web.config parameters for Python, Node.js and Go apps	(Optional) A standard Web.config will be generated and deployed to Azure App Service if the application does not have one. The values in web.config can be edited and vary based on the application framework. For example for node.js application, web.config will have startup file and iis_node module values. This edit feature is only for the generated web.config. Learn more.
App settings	(Optional) Edit web app application settings following the syntax -key value . Value containing spaces should be enclosed in double quotes. **Example** : -Port 5000 -RequestTimeout 5000 -WEBSITE_TIME_ZONE "Eastern Standard Time"
Configuration settings	(Optional) Edit web app configuration settings following the syntax -key value. Value containing spaces should be enclosed in double quotes. Example : -phpVersion 5.6 -linuxFxVersion: node|6.11
Take App Offline	(Optional) Select the option to take the Azure App Service offline by placing an app_offline.htm file in the root directory of the App Service before the sync operation begins. The file will be removed after the sync operation completes successfully.
Publish using Web Deploy	(Optional) Publish using Web Deploy options are supported only when using Windows agent. On other platforms, the task relies on Kudu REST APIs to deploy the Azure App Service, and following options are not supported
SetParameters file	(Optional) Optional: location of the SetParameters.xml file to use.

ARGUMENT	DESCRIPTION
Remove additional files at destination	(Optional) Select the option to delete files on the Azure App Service that have no matching files in the App Service package or folder. Note: This will also remove all files related to any extension installed on this Azure App Service. To prevent this, select 'Exclude files from App_Data folder' checkbox.
Exclude files from the App_Data folder	(Optional) Select the option to prevent files in the App_Data folder from being deployed to the Azure App Service.
Additional arguments	(Optional) Additional Web Deploy arguments following the syntax -key:value . These will be applied when deploying the Azure App Service. Example: -disableLink:AppPoolExtension -disableLink:ContentExtension. For more examples of Web Deploy operation settings, refer to this.
Rename locked files	(Optional) Select the option to enable msdeploy flag MSDEPLOY_RENAME_LOCKED_FILES=1 in Azure App Service application settings. The option if set enables msdeploy to rename locked files that are locked during app deployment
XML transformation	(Optional) The config transforms will be run for `*.Release.config` and `*.<EnvironmentName>.config` on the `*.config file`. Config transforms will be run prior to the Variable Substitution. XML transformations are supported only for Windows platform.
XML variable substitution	(Optional) Variables defined in the Build or Release Pipeline will be matched against the 'key' or 'name' entries in the appSettings, applicationSettings, and connectionStrings sections of any config file and parameters.xml. Variable Substitution is run after config transforms. Note: If same variables are defined in the Release Pipeline and in the Environment, then the Environment variables will supersede the Release Pipeline variables.

ARGUMENT	DESCRIPTION

JSON variable substitution	(Optional) Provide new line separated list of JSON files to substitute the variable values. Files names are to be provided relative to the root folder.
	To substitute JSON variables that are nested or hierarchical, specify them using JSONPath expressions.
	
	For example, to replace the value of 'ConnectionString' in the sample below, you need to define a variable as 'Data.DefaultConnection.ConnectionString' in the build/release pipeline (or release pipeline's environment).
	{
	"Data": {
	"DefaultConnection": {
	"ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True"
	}
	}
	}
	Variable Substitution is run after configuration transforms.
	
	Note: Build/release pipeline variables are excluded in substitution

CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Azure VM scale set Deployment

7/9/2018 • 2 minutes to read • Edit Online

Deploy Virtual Machine scale set image

YAML snippet

```
# Azure VM scale set Deployment
# Deploy Virtual Machine scale set image
- task: AzureVmssDeployment@0
  inputs:
    azureSubscription:
    #action: 'Update image' # Options: update Image, configure Application Startup
    vmssName:
    vmssOsType: # Options: windows, linux
    imageUrl:
    #customScriptsDirectory: # Optional
    #customScript: # Optional
    #customScriptArguments: # Optional
    #customScriptsStorageAccount: # Optional
    #skipArchivingCustomScripts: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription for the scale set.
Action	(Required) Choose between updating a VM scale set by using a VHD image and/or by running deployment/install scripts using Custom Script VM extension. The VHD image approach is better for scaling quickly and doing rollback. The extension approach is useful for post deployment configuration, software installation, or any other configuration / management task. You can use a VHD image to update a VM scale set only when it was created by using a custom image, the update will fail if the VM Scale set was created by using a platform/gallery image available in Azure. The Custom script VM extension approach can be used for VM scale set created by using either custom image or platform/gallery image.
Virtual Machine scale set name	(Required) Name of VM scale set which you want to update by using either a VHD image or by using Custom script VM extension.
OS type	(Required) Select the operating system type of VM scale set.
Image url	(Required) Specify the URL of VHD image. If it is an Azure storage blob url, the storage account location should be same as scale set location.

ARGUMENT	DESCRIPTION
Custom script directory	(Optional) Path to directory containing custom script(s) that will be run by using Custom Script VM extension. The extension approach is useful for post deployment configuration, application/software installation, or any other application configuration/management task. For example: the script can set a machine level environment variable which the application uses, like database connection string.
Command	(Optional) The script that will be run by using Custom Script VM extension. This script can invoke other scripts in the directory. The script will be invoked with arguments passed below. This script in conjugation with such arguments can be used to execute commands. For example: 1. Update-DatabaseConnectionStrings.ps1 -clusterType dev -user $(dbUser) -password $(dbUserPwd) will update connection string in web.config of web application. 2. install-secrets.sh --key-vault-type prod -key serviceprincipalkey will create an encrypted file containing service principal key.
Arguments	(Optional) The custom script will be invoked with arguments passed. Build/Release variables can be used which makes it easy to use secrets.
Azure storage account where custom scripts will be uploaded	(Optional) The Custom Script Extension downloads and executes scripts provided by you on each virtual machines in the VM scale set. These scripts will be stored in the storage account specified here. Specify a pre-existing ARM storage account.
Skip Archiving custom scripts	(Optional) By default, this task creates a compressed archive of directory containing custom scripts. This improves performance and reliability while uploading to azure storage. If not selected, archiving will not be done and all files will be inidividually uploaded.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Chef

7/9/2018 • 2 minutes to read • Edit Online

Deploy to Chef environments by editing environment attributes

YAML snippet

```
# Chef
# Deploy to Chef environments by editing environment attributes
- task: Chef@1
  inputs:
    connectedServiceName:
    environment:
    attributes:
    #chefWaitTime: '30'
```

Arguments

ARGUMENT	DESCRIPTION
Chef Connection	(Required) Name of the Chef subscription
Environment	(Required) Name of the Chef Environment to be used for Deployment. The attributes of that environment will be edited.
Environment Attributes	(Required) Specify the value of the leaf node attribute(s) to be updated. Example. { "default_attributes.connectionString" : "$(connectionString)", "override_attributes.buildLocation" : "https://sample.blob.core.windows.net/build" }. Task fails if the leaf node does not exist.
Wait Time	(Required) The amount of time (in minutes) to wait for this task to complete. Default value: 30 minutes
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Chef Knife

7/9/2018 • 2 minutes to read • Edit Online

Run Scripts with knife commands on your chef workstation

YAML snippet

```
# Chef Knife
# Run Scripts with knife commands on your chef workstation
- task: ChefKnife@1
  inputs:
    connectedServiceName:
    scriptPath:
    #scriptArguments: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Chef Subscription	(Required) Chef subscription to configure before running knife commands
Script Path	(Required) Path of the script. Should be fully qualified path or relative to the default working directory.
Script Arguments	(Optional) Additional parameters to pass to Script. Can be either ordinal or named parameters.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Copy Files Over SSH

7/10/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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. In addition to Linux, macOS is partially supported (see Q&A).

Prerequisites

- The task supports use of an SSH key pair to connect to the remote machine(s).
- The public key must be pre-installed or copied to the remote machine(s).

YAML snippet

```
# Copy Files Over SSH
# Copy files or build artifacts to a remote machine over SSH
- task: CopyFilesOverSSH@0
  inputs:
    sshEndpoint:
    #sourceFolder: # Optional
    #contents: '**'
    #targetFolder: # Optional
    #cleanTargetFolder: false # Optional
    #overwrite: true # Optional
    #failOnEmptySource: false # Optional
    #flattenFolders: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
SSH endpoint	The name of an SSH service connection containing connection details for the remote machine. - The hostname or IP address of the remote machine, the port number, and the user name are required to create an SSH service connection. - The private key and the passphrase must be specified for authentication.

ARGUMENT	DESCRIPTION
Source folder	The source folder for the files to copy to the remote machine. If omitted, the root of the repository is used. Names containing wildcards such as `*.zip` are not supported. Use variables if files are not in the repository. Example: `$(Agent.BuildDirectory)`
Contents	File paths to include as part of the copy. Supports multiple lines of minimatch patterns. Default is `**` which includes all files (including sub folders) under the source folder. - Example: `**/*.jar \n **/*.war` includes all jar and war files (including sub folders) under the source folder. - Example: `** \n !**/*.xml` includes all files (including sub folders) under the source folder but excludes xml files.
Target folder	Target folder on the remote machine to where files will be copied. Example: `/home/user/MySite`. Preface with a tilde (~) to specify the user's home directory.
Advanced - Clean target folder	If this option is selected, all existing files in the target folder will be deleted before copying.
Advanced - Overwrite	If this option is selected (the default), existing files in the target folder will be replaced.
Advanced - Flatten folders	If this option is selected, the folder structure is not preserved and all the files will be copied into the specified target folder on the remote machine.
Control options	See Control options

See also

- Install SSH Key task
- SSH task
- Blog post SSH build task

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Is this task supported for target machines running operating systems other than Linux?

This task is intended for target machines running Linux.

- For copying files to a macOS machine, this task may be used, but authenticating with a password is not

supported.

- For copying files to a Windows machine, consider using Windows Machine File Copy.

I use TFS on-premises and I don't see some of these features. Why not?

Some of these features are available only on 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.

Build: Docker

7/9/2018 • 3 minutes to read • Edit Online

Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure Container registry.

YAML snippet

```
# Docker
# Build, tag, push, or run Docker images, or run a Docker command. Task can be used with Docker or Azure
Container registry.
- task: Docker@1
  inputs:
    #containerregistrytype: 'Azure Container Registry' # Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscriptionEndpoint: # Optional
    #azureContainerRegistry: # Optional
    #command: 'Build an image' # Options: build An Image, tag Image, push An Image, run An Image
    #dockerFile: '**/Dockerfile' # Required when command == Build An Image
    #arguments: # Optional
    #useDefaultContext: true # Optional
    #buildContext: # Optional
    #pushMultipleImages: false # Optional
    #tagMultipleImages: false # Optional
    #imageName: '$(Build.Repository.Name):$(Build.BuildId)' # Required when command == Build An Image ||
Command == Run An Image || PushMultipleImages == False || TagMultipleImages == False
    #imageNamesPath: # Required when tagMultipleImages == True || PushMultipleImages == True
    #qualifyImageName: true # Optional
    #includeSourceTags: false # Optional
    #includeLatestTag: false # Optional
    #imageDigestFile: # Optional
    #containerName: # Optional
    #ports: # Optional
    #volumes: # Optional
    #envVars: # Optional
    #workingDirectory: # Optional
    #entrypointOverride: # Optional
    #containerCommand: # Optional
    #runInBackground: true # Optional
    #restartPolicy: 'no' # Required when command == Run An Image && RunInBackground == True# Options: no,
onFailure, always, unlessStopped
    #maxRestartRetries: # Optional
    #dockerHostEndpoint: # Optional
    #enforceDockerNamingConvention: true # Optional
    #memoryLimit: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.

ARGUMENT	DESCRIPTION
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Action	(Required) Select a Docker action.
Docker File	(Required) Path to the Docker file to use. Must be within the Docker build context.
Build Arguments	(Optional) Build-time variables for the Docker file. Specify each name=value pair on a new line.
Use Default Build Context	(Optional) Set the build context to the directory that contains the Docker file.
Build Context	(Optional) Path to the build context.
Image Name	(Required) Name of the Docker image to build, push, or run.
Image Names Path	(Required) Path to a text file that contains the names of the Docker images to tag or push. Each image name is contained on its own line.
Qualify Image Name	(Optional) Qualify the image name with the Docker registry connection's hostname if not otherwise specified.
Additional Image Tags	(Optional) Additional tags for the Docker image being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing the Docker image.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing the Docker image.
Image Digest File	(Optional) Path to a file that is created and populated with the full image repository digest of the Docker image that was pushed.
Container Name	(Optional) Name of the Docker container to run.
Ports	(Optional) Ports in the Docker container to publish to the host. Specify each host-port:container-port binding on a new line.
Volumes	(Optional) Volumes to mount from the host. Specify each host-dir:container-dir on a new line.
Environment Variables	(Optional) Environment variables for the Docker container. Specify each name=value pair on a new line.
Working Directory	(Optional) The working directory for the Docker container.

ARGUMENT	DESCRIPTION
Entrypoint Override	(Optional) Override the default entrypoint for the Docker container.
Command	(Optional) Command to run in the Docker container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the Docker container in the background.
Restart Policy	(Required) Select a restart policy.
Maximum Restart Retries	(Optional) The maximum number of restart retries the Docker daemon attempts.
Command	(Required) Docker command to execute, with arguments. For example, 'rmi -f image-name' to force remove an image.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
Force image name to follow Docker naming convention	(Optional) If enabled docker image name will be modified to follow Docker naming convention. Converts upper case character to lower case and removes spaces in image name.
Working Directory	(Optional) Working directory for the Docker command.
Memory limit	(Optional) The maximum amount of memory available to the container as a integer with optional suffixes like '2GB'.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Build: Docker Compose

7/9/2018 • 3 minutes to read • Edit Online

Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container registry.

YAML snippet

```
# Docker Compose
# Build, push or run multi-container Docker applications. Task can be used with Docker or Azure Container
registry.
- task: DockerCompose@0
  inputs:
    #containerregistrytype: 'Azure Container Registry' # Options: azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscription: # Optional
    #azureContainerRegistry: # Optional
    #dockerComposeFile: '**/docker-compose.yml'
    #additionalDockerComposeFiles: # Optional
    #dockerComposeFileArgs: # Optional
    #projectName: '$(Build.Repository.Name)' # Optional
    #qualifyImageNames: true # Optional
    #action: 'Run a Docker Compose command' # Options: build Services, push Services, run Services, run A
Specific Service, lock Services, write Service Image Digests, combine Configuration, run A Docker Compose
Command
    #additionalImageTags: # Optional
    #includeSourceTags: false # Optional
    #includeLatestTag: false # Optional
    #buildImages: true # Optional
    #serviceName: # Required when action == Run A Specific Service
    #containerName: # Optional
    #ports: # Optional
    #workingDirectory: # Optional
    #entrypoint: # Optional
    #containerCommand: # Optional
    #detached: true # Optional
    #abortOnContainerExit: true # Optional
    #imageDigestComposeFile: '$(Build.StagingDirectory)/docker-compose.images.yml' # Required when action ==
Write Service Image Digests
    #removeBuildOptions: false # Optional
    #baseResolveDirectory: # Optional
    #outputDockerComposeFile: '$(Build.StagingDirectory)/docker-compose.yml' # Required when action == Lock
Services || Action == Combine Configuration
    #dockerComposeCommand: # Required when action == Run A Docker Compose Command
    #dockerHostEndpoint: # Optional
    #nopIfNoDockerComposeFile: false # Optional
    #requireAdditionalDockerComposeFiles: false # Optional
    #currentWorkingDirectory: '$(System.DefaultWorkingDirectory)' # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Container Registry Type	(Required) Select a Container Registry Type.

ARGUMENT	DESCRIPTION
Docker Registry Connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select an Azure subscription
Azure Container Registry	(Optional) Select an Azure Container Registry
Docker Compose File	(Required) Path to the primary Docker Compose file to use.
Additional Docker Compose Files	(Optional) Additional Docker Compose files to be combined with the primary Docker Compose file. Relative paths are resolved relative to the directory containing the primary Docker Compose file. If a specified file is not found, it is ignored. Specify each file path on a new line.
Environment Variables	(Optional) Environment variables to be set during the command. Specify each name=value pair on a new line.
Project Name	(Optional) Project name used for default naming of images and containers.
Qualify Image Names	(Optional) Qualify image names for built services with the Docker registry connection's hostname if not otherwise specified.
Action	(Required) Select a Docker Compose action.
Additional Image Tags	(Optional) Additional tags for the Docker images being built or pushed.
Include Source Tags	(Optional) Include Git tags when building or pushing Docker images.
Include Latest Tag	(Optional) Include the 'latest' tag when building or pushing Docker images.
Build Images	(Optional) Build images before starting service containers.
Service Name	(Required) Name of the specific service to run.
Container Name	(Optional) Name of the specific service container to run.
Ports	(Optional) Ports in the specific service container to publish to the host. Specify each host-port:container-port binding on a new line.
Working Directory	(Optional) The working directory for the specific service container.
Entrypoint Override	(Optional) Override the default entry point for the specific service container.

ARGUMENT	DESCRIPTION
Command	(Optional) Command to run in the specific service container. For example, if the image contains a simple Python Flask web application you can specify 'python app.py' to launch the web application.
Run In Background	(Optional) Run the service containers in the background.
Abort on Container Exit	(Optional) Stop all containers when any container exits.
Image Digest Compose File	(Required) Path to a Docker Compose file that is created and populated with the full image repository digests of each service's Docker image.
Remove Build Options	(Optional) Remove the build options from the output Docker Compose file.
Base Resolve Directory	(Optional) The base directory from which relative paths in the output Docker Compose file should be resolved.
Output Docker Compose File	(Required) Path to an output Docker Compose file.
Command	(Required) Docker Compose command to execute with arguments. For example, 'rm --all' to remove all stopped service containers.
Docker Host Connection	(Optional) Select a Docker host connection. Defaults to the agent's host.
No-op if no Docker Compose File	(Optional) If the Docker Compose file does not exist, skip this step. This is useful when the step offers optional behavior based on the existence of a Docker Compose file in the repository.
Require Additional Docker Compose Files	(Optional) Produces an error if the additional Docker Compose files do not exist. This overrides the default behavior which is to ignore a file if it does not exist.
Working Directory	(Optional) Working directory for the Docker Compose command.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Package and deploy Helm Charts

7/9/2018 • 3 minutes to read • Edit Online

Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.

YAML snippet

```
# Package and deploy Helm charts
# Deploy, configure, update your Kubernetes cluster in Azure Container Service by running helm commands.
- task: HelmDeploy@0
  inputs:
    #connectionType: 'Azure Resource Manager' # Options: azure Resource Manager, kubernetes Service Connection
    #azureSubscription: # Required when connectionType == Azure Resource Manager
    #azureResourceGroup: # Required when connectionType == Azure Resource Manager
    #kubernetesCluster: # Required when connectionType == Azure Resource Manager
    #kubernetesServiceConnection: # Required when connectionType == Kubernetes Service Connection
    #namespace: # Optional
    #command: 'ls' # Options: create, delete, expose, get, init, install, ls, package, rollback, upgrade
    #chartType: 'Name' # Required when command == Install || Command == Upgrade# Options: name, filePath
    #chartName: # Required when chartType == Name
    #chartPath: # Required when chartType == FilePath || Command == Package
    #chartVersion: # Optional
    #releaseName: # Optional
    #overrideValues: # Optional
    #valueFile: # Optional
    #destination: '$(Build.ArtifactStagingDirectory)' # Optional
    #canaryImage: false # Optional
    #upgradeTiller: true # Optional
    #updateDependency: false # Optional
    #save: true # Optional
    #install: true # Optional
    #recreate: false # Optional
    #resetValues: false # Optional
    #force: false # Optional
    #waitForExecution: true # Optional
    #arguments: # Optional
    #enableTls: false # Optional
    #caCert: # Required when enableTls == True
    #certificate: # Required when enableTls == True
    #privatekey: # Required when enableTls == True
    #tillerNamespace: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Connection Type	(Required) Select a Connection Type.
Azure subscription	(Required) Select an Azure subscription, which has your Azure Container Registry.
Resource group	(Required) Select an Azure Resource Group.
Kubernetes cluster	(Required) Select an Azure Managed Cluster.

ARGUMENT	DESCRIPTION
Kubernetes Service Connection	(Required) Select a Kubernetes service connection.
Namespace	(Optional) Specify K8 namespace to use. Use Tiller namespace in advance section to specify tiller namespace.
Command	(Required) Select a helm command.
Chart Type	(Required) Select how you want to enter chart info. You can either provide name of the chart or folder/file path to the chart.
Chart Name	(Required) Name of the chart to install or upgrade.
Chart Path	(Required) Path to the chart to install or upgrade.
Version	(Optional) Specify the exact chart version to install. If this is not specified, the latest version is installed. Set the version on the chart to this semver version
Release Name	(Optional) Release name. If unspecified, it will autogenerate one for you.
Set Values	(Optional) Set values on the command line (can specify multiple or separate values with commas: key1=val1,key2=val2).
Value File	(Optional) Specify values in a YAML file or a URL.
Destination	(Optional) Specify values in a YAML file or a URL.
Use canary image version.	(Optional) Use the canary Tiller image. Will install the latest pre-release version of Tiller.
Upgrade Tiller	(Optional) Upgrade if Tiller is already installed.
Update Dependency	(Optional) Run helm dependency update before installing the chart. Update dependencies from 'requirements.yaml' to dir 'charts/' before packaging
Save	(Optional) Save packaged chart to local chart repository (default true)
Install if release not present.	(Optional) If a release by this name doesn't already exist, run an install.
Recreate Pods.	(Optional) Performs pods restart for the resource if applicable.
Reset Values.	(Optional) Reset the values to the ones built into the chart.
Force	(Optional) Force resource update through delete/recreate if needed
Wait	(Optional) Block till command execution completes.

ARGUMENT	DESCRIPTION
Arguments	(Optional) Command arguments.
Enable TLS	(Optional) Enables using SSL between Helm and Tiller.
CA certificate	(Required) CA cert used to issue certificate for tiller and helm client.
Certificate	(Required) Specify Tiller certificate or Helm client certificate
Key	(Required) Specify Tiller Key or Helm client key
Tiller namespace	(Optional) Specify K8 namespace of tiller.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: IIS Web App Deploy

7/17/2018 • 2 minutes to read • Edit Online

Deploy a Website or Web Application using WebDeploy

YAML snippet

```
# IIS Web App Deploy
# Deploy a Website or Web Application using WebDeploy
- task: IISWebAppDeploymentOnMachineGroup@0
  inputs:
    webSiteName:
    #virtualApplication: # Optional
    #package: '$(System.DefaultWorkingDirectory)\**\*.zip'
    #setParametersFile: # Optional
    #removeAdditionalFilesFlag: false # Optional
    #excludeFilesFromAppDataFlag: false # Optional
    #takeAppOfflineFlag: false # Optional
    #additionalArguments: # Optional
    #xmlTransformation: # Optional
    #xmlVariableSubstitution: # Optional
    #jSONFiles: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Website Name	(Required) Provide the name of an existing website on the machine group machines
Virtual Application	(Optional) Specify the name of an already existing Virtual Application on the target Machines
Package or Folder	(Required) File path to the package or a folder generated by MSBuild or a compressed archive file. Variables (Build | Release), wild cards are supported. For example, $(System.DefaultWorkingDirectory)***.zip.
SetParameters File	(Optional) Optional: location of the SetParameters.xml file to use.
Remove Additional Files at Destination	(Optional) Select the option to delete files on the Web App that have no matching files in the Web App zip package.
Exclude Files from the App_Data Folder	(Optional) Select the option to prevent files in the App_Data folder from being deployed to the Web App.
Take App Offline	(Optional) Select the option to take the Web App offline by placing an app_offline.htm file in the root directory of the Web App before the sync operation begins. The file will be removed after the sync operation completes successfully.

ARGUMENT	DESCRIPTION
Additional Arguments	(Optional) Additional Web Deploy arguments that will be applied when deploying the Azure Web App like,-disableLink:AppPoolExtension -disableLink:ContentExtension.
XML transformation	(Optional) The config transforms will be run for `*.Release.config` and `*.<EnvironmentName>.config` on the `*.config file`. Config transforms will be run prior to the Variable Substitution. XML transformations are supported only for Windows platform.
XML variable substitution	(Optional) Variables defined in the Build or Release Pipeline will be matched against the 'key' or 'name' entries in the appSettings, applicationSettings, and connectionStrings sections of any config file and parameters.xml. Variable Substitution is run after config transforms. Note: If same variables are defined in the Release Pipeline and in the Environment, then the Environment variables will supersede the Release Pipeline variables.

JSON variable substitution	(Optional) Provide new line separated list of JSON files to substitute the variable values. Files names are to be provided relative to the root folder.
	To substitute JSON variables that are nested or hierarchical, specify them using JSONPath expressions.
	
	For example, to replace the value of 'ConnectionString' in the sample below, you need to define a variable as 'Data.DefaultConnection.ConnectionString' in the build/release pipeline (or release pipeline's environment).
	{
	"Data": {
	"DefaultConnection": {
	"ConnectionString": "Server=(localdb)\SQLEXPRESS;Database=MyDB;Trusted_Connection=True"
	}
	}
	}
	Variable Substitution is run after configuration transforms.
	
	Note: Build/release pipeline variables are excluded in substitution

CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: IIS Web App Manage

7/9/2018 • 9 minutes to read • Edit Online

Create or update a Website, Web App, Virtual Directories, and Application Pool

YAML snippet

```
# IIS Web App Manage
# Create or update a Website, Web App, Virtual Directories, and Application Pool
- task: IISWebAppManagementOnMachineGroup@0
  inputs:
    #enableIIS: false # Optional
    #iISDeploymentType: 'IISWebsite' # Options: iISWebsite, iISWebApplication, iISVirtualDirectory,
iISApplicationPool
    #actionIISWebsite: 'CreateOrUpdateWebsite' # Required when iISDeploymentType == IISWebsite# Options:
createOrUpdateWebsite, startWebsite, stopWebsite
    #actionIISApplicationPool: 'CreateOrUpdateAppPool' # Required when iISDeploymentType ==
IISApplicationPool# Options: createOrUpdateAppPool, startAppPool, stopAppPool, recycleAppPool
    #startStopWebsiteName: # Required when actionIISWebsite == StartWebsite || ActionIISWebsite == StopWebsite
    websiteName:
    #websitePhysicalPath: '%SystemDrive%\inetpub\wwwroot'
    #websitePhysicalPathAuth: 'WebsiteUserPassThrough' # Options: websiteUserPassThrough, websiteWindowsAuth
    #websiteAuthUserName: # Required when websitePhysicalPathAuth == WebsiteWindowsAuth
    #websiteAuthUserPassword: # Optional
    #addBinding: false # Optional
    #protocol: 'http' # Required when iISDeploymentType == RandomDeployment# Options: https, http
    #iPAddress: 'All Unassigned' # Required when iISDeploymentType == RandomDeployment
    #port: '80' # Required when iISDeploymentType == RandomDeployment
    #serverNameIndication: false # Optional
    #hostNameWithOutSNI: # Optional
    #hostNameWithHttp: # Optional
    #hostNameWithSNI: # Required when iISDeploymentType == RandomDeployment
    #sSLCertThumbPrint: # Required when iISDeploymentType == RandomDeployment
    bindings:
    #createOrUpdateAppPoolForWebsite: false # Optional
    #configureAuthenticationForWebsite: false # Optional
    appPoolNameForWebsite:
    #dotNetVersionForWebsite: 'v4.0' # Options: v4.0, v2.0, no Managed Code
    #pipeLineModeForWebsite: 'Integrated' # Options: integrated, classic
    #appPoolIdentityForWebsite: 'ApplicationPoolIdentity' # Options: applicationPoolIdentity, localService,
localSystem, networkService, specificUser
    #appPoolUsernameForWebsite: # Required when appPoolIdentityForWebsite == SpecificUser
    #appPoolPasswordForWebsite: # Optional
    #anonymousAuthenticationForWebsite: false # Optional
    #basicAuthenticationForWebsite: false # Optional
    #windowsAuthenticationForWebsite: true # Optional
    parentWebsiteNameForVD:
    virtualPathForVD:
    #physicalPathForVD: '%SystemDrive%\inetpub\wwwroot'
    #vDPhysicalPathAuth: 'VDUserPassThrough' # Optional. Options: vDUserPassThrough, vDWindowsAuth
    #vDAuthUserName: # Required when vDPhysicalPathAuth == VDWindowsAuth
    #vDAuthUserPassword: # Optional
    parentWebsiteNameForApplication:
    virtualPathForApplication:
    #physicalPathForApplication: '%SystemDrive%\inetpub\wwwroot'
    #applicationPhysicalPathAuth: 'ApplicationUserPassThrough' # Optional. Options:
applicationUserPassThrough, applicationWindowsAuth
    #applicationAuthUserName: # Required when applicationPhysicalPathAuth == ApplicationWindowsAuth
    #applicationAuthUserPassword: # Optional
    #createOrUpdateAppPoolForApplication: false # Optional
    appPoolNameForApplication:
```

```
    appPoolNameForApplication:
    #dotNetVersionForApplication: 'v4.0' # Options: v4.0, v2.0, no Managed Code
    #pipeLineModeForApplication: 'Integrated' # Options: integrated, classic
    #appPoolIdentityForApplication: 'ApplicationPoolIdentity' # Options: applicationPoolIdentity,
localService, localSystem, networkService, specificUser
    #appPoolUsernameForApplication: # Required when appPoolIdentityForApplication == SpecificUser
    #appPoolPasswordForApplication: # Optional
    appPoolName:
    #dotNetVersion: 'v4.0' # Options: v4.0, v2.0, no Managed Code
    #pipeLineMode: 'Integrated' # Options: integrated, classic
    #appPoolIdentity: 'ApplicationPoolIdentity' # Options: applicationPoolIdentity, localService, localSystem,
networkService, specificUser
    #appPoolUsername: # Required when appPoolIdentity == SpecificUser
    #appPoolPassword: # Optional
    #startStopRecycleAppPoolName: # Required when actionIISApplicationPool == StartAppPool ||
ActionIISApplicationPool == StopAppPool || ActionIISApplicationPool == RecycleAppPool
    #appCmdCommands: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Enable IIS	(Optional) Check this if you want to install IIS on the machine.
Configuration type	(Required) You can create or update sites, applications, virtual directories, and application pools.
Action	(Required) Select the appropriate action that you want to perform on an IIS website. "Create Or Update" will create a website or update an existing website. Start, Stop will start or stop the website respectively.
Action	(Required) Select the appropriate action that you want to perform on an IIS Application Pool. "Create Or Update" will create app-pool or update an existing one. Start, Stop, Recycle will start, stop or recycle the application pool respectively.
Website name	(Required) Provide the name of the IIS website.
Website name	(Required) Provide the name of the IIS website to create or update.
Physical path	(Required) Provide the physical path where the website content will be stored. The content can reside on the local Computer, or in a remote directory, or on a network share, like C:\Fabrikam or \\ContentShare\Fabrikam.
Physical path authentication	(Required) Select the authentication mechanism that will be used to access the physical path of the website.
Username	(Required) Provide the user name that will be used to access the website's physical path.

ARGUMENT	DESCRIPTION
Password	(Optional) Provide the user's password that will be used to access the website's physical path. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Add binding	(Optional) Select the option to add port binding for the website.
Protocol	(Required) Select HTTP for the website to have an HTTP binding, or select HTTPS for the website to have a Secure Sockets Layer (SSL) binding.
IP address	(Required) Provide an IP address that end-users can use to access this website. If 'All Unassigned' is selected, then the website will respond to requests for all IP addresses on the port and for the host name, unless another website on the server has a binding on the same port but with a specific IP address.
Port	(Required) Provide the port, where the Hypertext Transfer Protocol Stack (HTTP.sys) will listen to the website requests.
Server Name Indication required	(Optional) Select the option to set the Server Name Indication (SNI) for the website. SNI extends the SSL and TLS protocols to indicate the host name that the clients are attempting to connect to. It allows, multiple secure websites with different certificates, to use the same IP address.
Host name	(Optional) Enter a host name (or domain name) for the website. If a host name is specified, then the clients must use the host name instead of the IP address to access the website.
Host name	(Optional) Enter a host name (or domain name) for the website. If a host name is specified, then the clients must use the host name instead of the IP address to access the website.
Host name	(Required) Enter a host name (or domain name) for the website. If a host name is specified, then the clients must use the host name instead of the IP address to access the website.
SSL certificate thumbprint	(Required) Provide the thumb-print of the Secure Socket Layer certificate that the website is going to use for the HTTPS communication as a 40 character long hexadecimal string. The SSL certificate should be already installed on the Computer, at Local Computer, Personal store.
Add bindings	(Required) Click on the extension [...] button to add bindings for the website.

ARGUMENT	DESCRIPTION
Create or update app pool	(Optional) Select the option to create or update an application pool. If checked, the website will be created in the specified app pool.
Configure authentication	(Optional) Select the option to configure authentication for website.
Name	(Required) Provide the name of the IIS application pool to create or update.
.NET version	(Required) Select the version of the .NET Framework that is loaded by the application pool. If the applications assigned to this application pool do not contain managed code, then select the 'No Managed Code' option from the list.
Managed pipeline mode	(Required) Select the managed pipeline mode that specifies how IIS processes requests for managed content. Use classic mode only when the applications in the application pool cannot run in the Integrated mode.
Identity	(Required) Configure the account under which an application pool's worker process runs. Select one of the predefined security accounts or configure a custom account.
Username	(Required) Provide the username of the custom account that you want to use.
Password	(Optional) Provide the password for custom account. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Anonymous authentication	(Optional) Select the option to enable anonymous authentication for website.
Basic authentication	(Optional) Select the option to enable basic authentication for website.
Windows authentication	(Optional) Select the option to enable windows authentication for website.
Parent website name	(Required) Provide the name of the parent Website of the virtual directory.
Virtual path	(Required) Provide the virtual path of the virtual directory. Example: To create a virtual directory Site/Application/VDir enter /Application/Vdir. The parent website and application should be already existing.

ARGUMENT	DESCRIPTION
Physical path	(Required) Provide the physical path where the virtual directory's content will be stored. The content can reside on the local Computer, or in a remote directory, or on a network share, like C:\Fabrikam or \\ContentShare\Fabrikam.
Physical path authentication	(Optional) Select the authentication mechanism that will be used to access the physical path of the virtual directory.
Username	(Required) Provide the user name that will be used to access the virtual directory's physical path.
Password	(Optional) Provide the user's password that will be used to access the virtual directory's physical path. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Parent website name	(Required) Provide the name of the parent Website under which the application will be created or updated.
Virtual path	(Required) Provide the virtual path of the application. Example: To create an application Site/Application enter /Application. The parent website should be already existing.
Physical path	(Required) Provide the physical path where the application's content will be stored. The content can reside on the local Computer, or in a remote directory, or on a network share, like C:\Fabrikam or \\ContentShare\Fabrikam.
Physical path authentication	(Optional) Select the authentication mechanism that will be used to access the physical path of the application.
Username	(Required) Provide the user name that will be used to access the application's physical path.
Password	(Optional) Provide the user's password that will be used to access the application's physical path. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Create or update app pool	(Optional) Select the option to create or update an application pool. If checked, the application will be created in the specified app pool.
Name	(Required) Provide the name of the IIS application pool to create or update.

ARGUMENT	DESCRIPTION
.NET version	(Required) Select the version of the .NET Framework that is loaded by the application pool. If the applications assigned to this application pool do not contain managed code, then select the 'No Managed Code' option from the list.
Managed pipeline mode	(Required) Select the managed pipeline mode that specifies how IIS processes requests for managed content. Use classic mode only when the applications in the application pool cannot run in the Integrated mode.
Identity	(Required) Configure the account under which an application pool's worker process runs. Select one of the predefined security accounts or configure a custom account.
Username	(Required) Provide the username of the custom account that you want to use.
Password	(Optional) Provide the password for custom account. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Name	(Required) Provide the name of the IIS application pool to create or update.
.NET version	(Required) Select the version of the .NET Framework that is loaded by the application pool. If the applications assigned to this application pool do not contain managed code, then select the 'No Managed Code' option from the list.
Managed pipeline mode	(Required) Select the managed pipeline mode that specifies how IIS processes requests for managed content. Use classic mode only when the applications in the application pool cannot run in the Integrated mode.
Identity	(Required) Configure the account under which an application pool's worker process runs. Select one of the predefined security accounts or configure a custom account.
Username	(Required) Provide the username of the custom account that you want to use.
Password	(Optional) Provide the password for custom account. The best practice is to create a variable in the Build or Release pipeline, and mark it as 'Secret' to secure it, and then use it here, like '$(userCredentials)'. Note: Special characters in password are interpreted as per command-line arguments
Application pool name	(Required) Provide the name of the IIS application pool.

ARGUMENT	DESCRIPTION
Additional appcmd.exe commands	(Optional) Enter additional AppCmd.exe commands. For more than one command use a line separator, like list apppools list sites recycle apppool /apppool.name:ExampleAppPoolName
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Deploy to Kubernetes

7/9/2018 • 3 minutes to read • Edit Online

Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.

YAML snippet

```
# Deploy to Kubernetes
# Deploy, configure, update your Kubernetes cluster in Azure Container Service by running kubectl commands.
- task: Kubernetes@1
  inputs:
    #connectionType: 'Azure Resource Manager' # Options: azure Resource Manager, kubernetes Service Connection
    #kubernetesServiceEndpoint: # Required when connectionType == Kubernetes Service Connection
    #azureSubscriptionEndpoint: # Required when connectionType == Azure Resource Manager
    #azureResourceGroup: # Required when connectionType == Azure Resource Manager
    #kubernetesCluster: # Required when connectionType == Azure Resource Manager
    #namespace: # Optional
    #command: 'apply' # Options: apply, create, delete, exec, expose, get, logs, run, set, top
    #useConfigurationFile: false # Optional
    #configuration: # Required when useConfigurationFile == True
    #arguments: # Optional
    #secretType: 'dockerRegistry' # Options: dockerRegistry, generic
    #secretArguments: # Optional
    #containerRegistryType: 'Azure Container Registry' # Required when secretType == DockerRegistry# Options:
azure Container Registry, container Registry
    #dockerRegistryEndpoint: # Optional
    #azureSubscriptionEndpointForSecrets: # Optional
    #azureContainerRegistry: # Optional
    #secretName: # Optional
    #forceUpdate: true # Optional
    #configMapName: # Optional
    #forceUpdateConfigMap: false # Optional
    #useConfigMapFile: false # Optional
    #configMapFile: # Required when useConfigMapFile == True
    #configMapArguments: # Optional
    #versionOrLocation: 'version' # Optional. Options: version, location
    #versionSpec: '1.7.0' # Optional
    #checkLatest: false # Optional
    #specifyLocation: # Required when versionOrLocation == Location
    #workingDirectory: '$(System.DefaultWorkingDirectory)' # Optional
    #outputFormat: 'json' # Optional. Options: json, yaml
```

Arguments

ARGUMENT	DESCRIPTION
Kubernetes service connection	(Optional) Select a Kubernetes service connection.
Namespace	(Optional) Set the namespace for the kubectl command by using the –namespace flag. If the namespace is not provided, the commands will run in the default namespace.
Command	(Required) Select or specify a kubectl command to run.

ARGUMENT	DESCRIPTION
Use Configuration files	(Optional) Use Kubernetes configuration file with the kubectl command. Filename, directory, or URL to Kubernetes configuration files can also be provided.
Configuration file	(Required) Filename, directory, or URL to kubernetes configuration files that will be used with the commands.
Arguments	(Optional) Arguments to the specified kubectl command.
Type of secret	(Required) Create/update a generic or dockerimagepullsecret.
Arguments	(Optional) Specify keys and literal values to insert in secret.For example, --from-literal=key1=value1 --from-literal=key2="top secret".
Container Registry type	(Required) Select a Container registry type. The task can use Azure Subscription details to work with an Azure Container registry. Other standard Container registries are also supported.
Docker Registry connection	(Optional) Select a Docker registry connection. Required for commands that need to authenticate with a registry.
Azure subscription	(Optional) Select the Azure Resource Manager subscription, which contains Azure Container Registry. Note: To configure new service connection, select the Azure subscription from the list and click 'Authorize'. If your subscription is not listed or if you want to use an existing Service Principal, you can setup an Azure service connection using 'Add' or 'Manage' button.
Azure Container Registry	(Optional) Select an Azure Container Registry which will be used for pulling container images and deploying applications to the Kubernetes cluster. Required for commands that need to authenticate with a registry.
Secret name	(Optional) Name of the secret. You can use this secret name in the Kubernetes YAML configuration file.
Force update secret	(Optional) Delete the secret if it exists and create a new one with updated values.
ConfigMap name	(Optional) ConfigMaps allow you to decouple configuration artifacts from image content to keep containerized applications portable.
Force update configmap	(Optional) Delete the configmap if it exists and create a new one with updated values.
Arguments	(Optional) Specify keys and literal values to insert in configMap.For example, --from-literal=key1=value1 --from-literal=key2="top secret".
Kubectl	(Optional) undefined

ARGUMENT	DESCRIPTION
Version spec	(Optional) Version Spec of version to get. Examples: 1.7.0, 1.x.0, 4.x.0, 6.10.0, >=6.10.0
Check for latest version	(Optional) Always checks online for the latest available version (stable.txt) that satisfies the version spec. This is typically false unless you have a specific scenario to always get latest. This will cause it to incur download costs when potentially not necessary, especially with the hosted build pool.
Path to Kubectl	(Required) Full path to the kubectl.exe
Working directory	(Optional) Working directory for the Kubectl command.
Output format	(Optional) Output format.
Output variable name	(Optional) Name of the variable in which output of the command should be saved.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: Build Packer machine image

7/9/2018 • 3 minutes to read • Edit Online

Build machine image using Packer. This image can be used for Azure Virtual machine scale set deployment

YAML snippet

```
# Build Machine Image
# Build machine image using Packer. This image can be used for Azure Virtual machine scale set deployment
- task: PackerBuild@0
  inputs:
    #templateType: 'builtin' # Options: builtin, custom
    #customTemplateLocation: # Required when templateType == Custom
    #customTemplateParameters: '{}' # Optional
    connectedServiceName:
    location:
    storageAccountName:
    azureResourceGroup:
    #baseImageSource: 'default' # Options: default, customVhd
    #baseImage: 'MicrosoftWindowsServer:WindowsServer:2012-R2-Datacenter:windows' # Required when
baseImageSource == Default# Options: microsoftWindowsServer:WindowsServer:2012-R2-Datacenter:Windows,
microsoftWindowsServer:WindowsServer:2016-Datacenter:Windows, microsoftWindowsServer:WindowsServer:2012-
Datacenter:Windows, microsoftWindowsServer:WindowsServer:2008-R2-SP1:Windows, canonical:UbuntuServer:14.04.4-
LTS:Linux, canonical:UbuntuServer:16.04-LTS:Linux, redHat:RHEL:7.2:Linux, redHat:RHEL:6.8:Linux,
openLogic:CentOS:7.2:Linux, openLogic:CentOS:6.8:Linux, credativ:Debian:8:Linux, credativ:Debian:7:Linux,
sUSE:OpenSUSE-Leap:42.2:Linux, sUSE:SLES:12-SP2:Linux, sUSE:SLES:11-SP4:Linux
    #customImageUrl: # Required when baseImageSource == CustomVhd
    #customImageOSType: 'windows' # Required when baseImageSource == CustomVhd# Options: windows, linux
    packagePath:
    deployScriptPath:
    #deployScriptArguments: # Optional
    #additionalBuilderParameters: '{}' # Optional
    #skipTempFileCleanupDuringVMDeprovision: # Optional
    #imageUri: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Packer template	(Required) Select whether you want the task to auto generate Packer template or use custom template provided by you.
Packer template location	(Required) Path to a custom user-provided template.
Template parameters	(Optional) Specify parameters which will be passed to Packer for building custom template. This should map to "variables" section in your custom template. E.g. if the template has a variable named "drop-location", then add a parameter here with name "drop-location" and a value which you want to use. You can link the value to a release variable as well. To view/edit the additional parameters in a grid, click on "..." next to text box.

ARGUMENT	DESCRIPTION
Azure subscription	(Required) Select the Azure Resource Manager subscription for baking and storing the machine image.
Storage location	(Required) Location for storing the built machine image. This location will also be used to create a temporary VM for the purpose of building image.
Storage account	(Required) Storage account for storing the built machine image. This storage account must be pre-existing in the location selected.
Resource group	(Required) Azure Resource group that contains the selected storage account.
Base image source	(Required) Select the source of base image. You can either choose from a curated gallery of OS images or provide url of your custom image.
Base image	(Required) Choose from curated list of OS images. This will be used for installing pre-requisite(s) and application(s) before capturing machine image.
Base image URL	(Required) Specify url of base image. This will be used for installing pre-requisitie(s) and application(s) before capturing machine image.
Base image OS	(Required) undefined
Deployment Package	(Required) Specify the path for deployment package directory relative to $(System.DefaultWorkingDirectory). Supports minimatch pattern. Example path: FrontendWebApp/**/GalleryApp**
Deployment script	(Required) Specify the relative path to powershell script(for Windows) or shell script(for Linux) which deploys the package. This script should be contained in the package path selected above. Supports minimatch pattern. Example path: deploy//scripts/windows/deploy.ps1
Deployment script arguments	(Optional) Specify the arguments to be passed to deployment script.
Additional Builder parameters	(Optional) In auto generated Packer template mode the task creates a Packer template with an Azure builder. This builder is used to generate a machine image. You can add keys to the Azure builder to customize the generated Packer template. For example setting ssh_tty=true in case you are using a CentOS base image and you need to have a tty to run sudo. To view/edit the additional parameters in a grid, click on "..." next to text box.
Skip temporary file cleanup during deprovision	(Optional) During deprovisioning of VM, skip clean-up of temporary files uploaded to VM. Refer here

ARGUMENT	DESCRIPTION
Image URL	(Optional) Provide a name for the output variable which will store generated machine image url.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: PowerShell on Target Machines

7/9/2018 • 5 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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.

YAML snippet

```
# PowerShell on Target Machines
# Execute PowerShell scripts on remote machine(s). This version of the task uses PSSession and Invoke-Command for
remoting.
- task: PowerShellOnTargetMachines@3
  inputs:
    machines:
    #userName: # Optional
    #userPassword: # Optional
    #scriptType: 'Inline' # Optional. Options: filePath, inline
    #scriptPath: # Required when scriptType == FilePath
    #inlineScript: '# Write your powershell commands here.Write-Output Hello World' # Required when scriptType ==
Inline
    #scriptArguments: # Optional
    #initializationScript: # Optional
    #sessionVariables: # Optional
    #communicationProtocol: 'Https' # Optional. Options: http, https
    #authenticationMechanism: 'Default' # Optional. Options: default, credssp
    #newPsSessionOptionArguments: '-SkipCACheck -IdleTimeout 7200000 -OperationTimeout 0 -OutputBufferingMode Block'
# Optional
    #errorActionPreference: 'stop' # Optional. Options: stop, continue, silentlyContinue
    #failOnStderr: false # Optional
    #ignoreLASTEXITCODE: false # Optional
    #workingDirectory: # Optional
    #runPowershellInParallel: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Machines	A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. Can be: - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: `dbserver.fabrikam.com,dbserver_int.fabrikam.com:5986,192.168.34:5` - An output variable from a previous task. If you do not specify a port, the default WinRM port is used. This depends on the protocol you have configured: for WinRM 2.0, the default HTTP port is 5985 and the default HTTPS port is 5986.
Admin Login	The username of either a domain or a local administrative account on the target host(s). - Formats such as **username**, **domain\username**, **machine-name\username**, and **.\username** are supported. - UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.
Password	The password for the administrative account specified above. Consider using a secret variable global to the build or release pipeline to hide the password. Example: `$(passwordVariable)`
Protocol	The protocol that will be used to connect to the target host, either **HTTP** or **HTTPS**.
Test Certificate	If you choose the **HTTPS** option, set this checkbox to skip validating the authenticity of the machine's certificate by a trusted certification authority.
Deployment - PowerShell Script	The location of the PowerShell script on the target machine. Can include environment variables such as `$env:windir` and `$env:systemroot` Example: `C:\FabrikamFibre\Web\deploy.ps1`

ARGUMENT	DESCRIPTION
Deployment - Script Arguments	The arguments required by the script, if any. Example: `-applicationPath $(applicationPath) -username $(vmusername) -password $(vmpassword)`
Deployment - Initialization Script	The location on the target machine(s) of the data script used by PowerShell-DSC. It is recommended to use arguments instead of an initialization script.
Deployment - Session Variables	Used to set up the session variables for the PowerShell scripts. A comma-separated list such as `$varx=valuex, $vary=valuey` Most commonly used for backward compatibility with earlier versions of Release Management. It is recommended to use arguments instead of session variables.
Advanced - Run PowerShell in Parallel	Set this option to execute the PowerShell scripts in parallel on all the target machines
Advanced - Select Machines By	Depending on how you want to specify the machines in the group when using the **Filter Criteria** parameter, choose **Machine Names** or **Tags**.
Advanced - Filter Criteria	Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be: - The name of an Azure Resource Group. - An output variable from a previous task. - A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses. Specify tag names for a filter as {TagName}**:**{Value} Example: `Role:DB;OS:Win8.1`
Control options	See Control options

> Version 3.x of the task includes the **Inline script** setting where you can enter your PowerShell script code.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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.

YAML snippet

```
# Service Fabric Application Deployment
# Deploy a Service Fabric application to a cluster.
- task: ServiceFabricDeploy@1
  inputs:
    applicationPackagePath:
    serviceConnectionName:
    #publishProfilePath: # Optional
    #applicationParameterPath: # Optional
    #overrideApplicationParameter: false # Optional
    #compressPackage: false # Optional
    #copyPackageTimeoutSec: # Optional
    #registerPackageTimeoutSec: # Optional
    #overwriteBehavior: 'SameAppTypeAndVersion' # Options: always, never, sameAppTypeAndVersion
    #skipUpgradeSameTypeAndVersion: false # Optional
    #skipPackageValidation: false # Optional
    #useDiffPackage: false # Optional
    #overridePublishProfileSettings: false # Optional
    #isUpgrade: true # Optional
    #unregisterUnusedVersions: true # Optional
    #upgradeMode: 'Monitored' # Required when overridePublishProfileSettings == True && IsUpgrade == True# Options: monitored,
unmonitoredAuto, unmonitoredManual
    #failureAction: 'Rollback' # Required when overridePublishProfileSettings == True && IsUpgrade == True && UpgradeMode == Monitored#
Options: rollback, manual
    #upgradeReplicaSetCheckTimeoutSec: # Optional
    #replicaQuorumTimeoutSec: # Optional
    #timeoutSec: # Optional
    #forceRestart: false # Optional
    #healthCheckRetryTimeoutSec: # Optional
    #healthCheckWaitDurationSec: # Optional
    #healthCheckStableDurationSec: # Optional
    #upgradeDomainTimeoutSec: # Optional
    #considerWarningAsError: false # Optional
    #defaultServiceTypeHealthPolicy: # Optional
    #maxPercentUnhealthyDeployedApplications: # Optional
    #upgradeTimeoutSec: # Optional
    #serviceTypeHealthPolicyMap: # Optional
    #configureDockerSettings: false # Optional
    #registryCredentials: 'AzureResourceManagerEndpoint' # Required when configureDockerSettings == True# Options:
azureResourceManagerEndpoint, containerRegistryEndpoint, usernamePassword
    #dockerRegistryConnection: # Required when configureDockerSettings == True && RegistryCredentials == ContainerRegistryEndpoint
    #azureSubscription: # Required when configureDockerSettings == True && RegistryCredentials == AzureResourceManagerEndpoint
    #registryUserName: # Optional
    #registryPassword: # Optional
    #passwordEncrypted: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Publish Profile	The location of the publish profile that specifies the settings to use for deployment, including the location of the target Service Fabric cluster. Can include wildcards and variables. Example: `$(system.defaultworkingdirectory)/**/drop/projectartifacts/**/PublishProfi`
Application Package	The location of the Service Fabric application package to be deployed to the cluster. Can include wildcards and variables. Example: `$(system.defaultworkingdirectory)/**/drop/applicationpackage`
Cluster Connection	The name of the Azure Service Fabric service connection defined in the TS/TFS project that describes the connection to the cluster.
Control options	See Control options

Also see: Update Service Fabric App Versions task

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/10/2018 • 2 minutes 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.

YAML snippet

```
# Service Fabric Compose Deploy
# Deploy a docker-compose application to a Service Fabric cluster.
- task: ServiceFabricComposeDeploy@0
  inputs:
    clusterConnection:
    #composeFilePath: '**/docker-compose.yml'
    #applicationName: 'fabric:/Application1'
    #registryCredentials: 'AzureResourceManagerEndpoint' # Options: azureResourceManagerEndpoint,
containerRegistryEndpoint, usernamePassword, none
    #dockerRegistryConnection: # Optional
    #azureSubscription: # Required when registryCredentials == AzureResourceManagerEndpoint
    #registryUserName: # Optional
    #registryPassword: # Optional
    #passwordEncrypted: # Optional
    #upgrade: # Optional
    #deployTimeoutSec: # Optional
    #removeTimeoutSec: # Optional
    #getStatusTimeoutSec: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Cluster Connection	The Azure Service Fabric service connection to use to connect and authenticate to the cluster.
Compose File Path	Path to the compose file that is to be deployed. Can include wildcards and variables. Example: `$(System.DefaultWorkingDirectory)/**/drop/projectartifacts/**/docl compose.yml` . **Note**: combining compose files is not supported as part of this task.
Application Name	The Service Fabric Application Name of the application being deployed. Use `fabric:/` as a prefix. Application Names within a Service Fabric cluster must be unique.

ARGUMENT	DESCRIPTION
Registry Credentials Source	Specifies how credentials for the Docker container registry will be provided to the deployment task: **Azure Resource Manager Endpoint**: An Azure Resource Manager service connection and Azure subscription to be used to obtain a service principal ID and key for an Azure Container Registry. **Container Registry Endpoint**: A Docker registry service connection. If a certificate matching the Server Certificate Thumbprint in the Cluster Connection is installed on the build agent, it will be used to encrypt the password; otherwise the password will not be encrypted and sent in clear text. **Username and Password**: Username and password to be used. We recommend you encrypt your password using `Invoke-ServiceFabricEncryptText` (Check **Password Encrypted**). If you do not, and a certificate matching the Server Certificate Thumbprint in the Cluster Connection is installed on the build agent, it will be used to encrypt the password; otherwise the password will not be encrypted and sent in clear text. **None**: No registry credentials are provided (used for accessing public container registries).
Deploy Timeout (s)	Timeout in seconds for deploying the application.
Remove Timeout (s)	Timeout in seconds for removing an existing application.
Get Status Timeout (s)	Timeout in seconds for getting the status of an existing application.
Control options	See Control options

Also see: Service Fabric PowerShell Utility

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

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 SQL Database Deployment

7/17/2018 • 3 minutes to read • Edit Online

Deploy Azure SQL DB using DACPAC or run scripts using SQLCMD

YAML snippet

```
# Azure SQL Database Deployment
# Deploy Azure SQL DB using DACPAC or run scripts using SQLCMD
- task: SqlAzureDacpacDeployment@1
  inputs:
    #azureConnectionType: 'ConnectedServiceNameARM' # Optional. Options: connectedServiceName,
connectedServiceNameARM
    #azureClassicSubscription: # Required when azureConnectionType == ConnectedServiceName
    #azureSubscription: # Required when azureConnectionType == ConnectedServiceNameARM
    serverName:
    databaseName:
    #sqlUsername: # Optional
    #sqlPassword: # Optional
    #deploymentAction: 'Publish' # Options: publish, extract, export, import, script, driftReport,
deployReport
    #taskNameSelector: 'DacpacTask' # Optional. Options: dacpacTask, sqlTask, inlineSqlTask
    #dacpacFile: # Required when taskNameSelector == DacpacTask || DeploymentAction == Script ||
DeploymentAction == DeployReport
    #bacpacFile: # Required when deploymentAction == Import
    #sqlFile: # Required when taskNameSelector == SqlTask
    #sqlInline: # Required when taskNameSelector == InlineSqlTask
    #publishProfile: # Optional
    #additionalArguments: # Optional
    #sqlAdditionalArguments: # Optional
    #inlineAdditionalArguments: # Optional
    #ipDetectionMethod: 'AutoDetect' # Options: autoDetect, iPAddressRange
    #startIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #endIpAddress: # Required when ipDetectionMethod == IPAddressRange
    #deleteFirewallRule: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Azure Connection Type	(Optional)
Azure Classic Subscription	(Required) Target Azure Classic subscription for deploying SQL files
Azure Subscription	(Required) Target Azure Resource Manager subscription for deploying SQL files
Azure SQL Server Name	(Required) Azure SQL Server name, like Fabrikam.database.windows.net,1433 or Fabrikam.database.windows.net.
Database Name	(Required) Name of the Azure SQL Database, where the files will be deployed.

ARGUMENT	DESCRIPTION
Server Admin Login	(Optional) Specify the Azure SQL Server administrator login.
Password	(Optional) Password for the Azure SQL Server administrator. It can accept variables defined in build/release pipelines as '$(passwordVariable)'. You may mark the variable type as 'secret' to secure it.
Type	(Optional)
DACPAC File	(Required) Location of the DACPAC file on the automation agent or on a UNC path accessible to the automation agent like, \\BudgetIT\Web\Deploy\FabrikamDB.dacpac. Predefined system variables like, $(agent.releaseDirectory) can also be used here.
SQL Script	(Required) Location of the SQL script file on the automation agent or on a UNC path accessible to the automation agent like, \\BudgetIT\Web\Deploy\FabrikamDB.sql. Predefined system variables like, $(agent.releaseDirectory) can also be used here.
Inline SQL Script	(Required) Enter the SQL script to execute on the Database selected above.
Publish Profile	(Optional) Publish profile provides fine-grained control over Azure SQL Database creation or upgrades. Specify the path to the Publish profile XML file on the automation agent or on a UNC share. Predefined system variables like, $(agent.buildDirectory) or $(agent.releaseDirectory) can also be used here.
Additional SqlPackage.exe Arguments	(Optional) Additional SqlPackage.exe arguments that will be applied when deploying the Azure SQL Database, in case DACPAC option is selected like, /p:IgnoreAnsiNulls=True /p:IgnoreComments=True. These arguments will override the settings in the Publish profile XML file (if provided).
Additional Invoke-Sqlcmd Arguments	(Optional) Additional Invoke-Sqlcmd arguments that will be applied when executing the given SQL query on the Azure SQL Database like, -ConnectionTimeout 100 -OutputSqlErrors.
Additional Invoke-Sqlcmd Arguments	(Optional) Additional Invoke-Sqlcmd arguments that will be applied when executing the given SQL query on the Azure SQL Database like, -ConnectionTimeout 100 -OutputSqlErrors
Specify Firewall Rules Using	(Required) For the task to run, the IP Address of the automation agent has to be added to the 'Allowed IP Addresses' in the Azure SQL Server's Firewall. Select auto-detect to automatically add firewall exception for range of possible IP Address of automation agent or specify the range explicitly.
Start IP Address	(Required) The starting IP Address of the automation agent machine pool like 196.21.30.50.

ARGUMENT	DESCRIPTION
End IP Address	(Required) The ending IP Address of the automation agent machine pool like 196.21.30.65.
Delete Rule After Task Ends	(Optional) If selected, then after the task ends, the IP Addresses specified here are deleted from the 'Allowed IP Addresses' list of the Azure SQL Server's Firewall.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: SQL Server Database Deploy

7/9/2018 • 3 minutes to read • Edit Online

Deploy to SQL Server Database using DACPAC or SQL scripts

YAML snippet

```
# SQL Server Database Deploy
# Deploy to SQL Server Database using DACPAC or SQL scripts
- task: SqlDacpacDeploymentOnMachineGroup@0
  inputs:
    #taskType: 'dacpac' # Options: dacpac, sqlQuery, sqlInline
    #dacpacFile: # Required when taskType == Dacpac
    #sqlFile: # Required when taskType == SqlQuery
    #executeInTransaction: false # Optional
    #exclusiveLock: false # Optional
    #appLockName: # Required when exclusiveLock == True
    #inlineSql: # Required when taskType == SqlInline
    #targetMethod: 'server' # Required when taskType == Dacpac# Options: server, connectionString,
publishProfile
    #serverName: 'localhost' # Required when targetMethod == Server || TaskType == SqlQuery || TaskType ==
SqlInline
    #databaseName: # Required when targetMethod == Server || TaskType == SqlQuery || TaskType == SqlInline
    #authScheme: 'windowsAuthentication' # Required when targetMethod == Server || TaskType == SqlQuery ||
TaskType == SqlInline# Options: windowsAuthentication, sqlServerAuthentication
    #sqlUsername: # Required when authScheme == SqlServerAuthentication
    #sqlPassword: # Required when authScheme == SqlServerAuthentication
    #connectionString: # Required when targetMethod == ConnectionString
    #publishProfile: # Optional
    #additionalArguments: # Optional
    #additionalArgumentsSql: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Deploy SQL Using	(Required) Specify the way in which you want to deploy DB, either by using Dacpac or by using Sql Script.
DACPAC File	(Required) Location of the DACPAC file on the target machines or on a UNC path like, \\BudgetIT\Web\Deploy\FabrikamDB.dacpac. The UNC path should be accessible to the machine's administrator account. Environment variables are also supported, like $env:windir, $env:systemroot, $env:windir\FabrikamFibre\DB. Wildcards can be used. For example, `**/*.dacpac` for DACPAC file present in all sub folders.

ARGUMENT	DESCRIPTION
Sql File	(Required) Location of the SQL file on the target. Provide semi-colon separated list of SQL script files to execute multiple files. The SQL scripts will be executed in the order given. Location can also be a UNC path like, \\BudgetIT\Web\Deploy\FabrikamDB.sql. The UNC path should be accessible to the machine's administrator account. Environment variables are also supported, like $env:windir, $env:systemroot, $env:windir\FabrikamFibre\DB. Wildcards can be used. For example, `**/*.sql` for sql file present in all sub folders.
Execute within a transaction	(Optional) Executes SQL script(s) within a transaction
Acquire an exclusive app lock while executing script(s)	(Optional) Acquires an exclusive app lock while executing script(s)
App lock name	(Required) App lock name
Inline Sql	(Required) Sql Queries inline
Specify SQL Using	(Required) Specify the option to connect to the target SQL Server Database. The options are either to provide the SQL Server Database details, or the SQL Server connection string, or the Publish profile XML file.
Server Name	(Required) Provide the SQL Server name like, machinename\FabriakmSQL,1433 or localhost or .\SQL2012R2. Specifying localhost will connect to the Default SQL Server instance on the machine.
Database Name	(Required) Provide the name of the SQL Server database.
Authentication	(Required) Select the authentication mode for connecting to the SQL Server. In Windows authentication mode, the administrator's account, as specified in the Machines section, is used to connect to the SQL Server. In SQL Server Authentication mode, the SQL login and Password have to be provided in the parameters below.
SQL User name	(Required) Provide the SQL login to connect to the SQL Server. The option is only available if SQL Server Authentication mode has been selected.
SQL Password	(Required) Provide the Password of the SQL login. The option is only available if SQL Server Authentication mode has been selected.
Connection String	(Required) Specify the SQL Server connection string like "Server=localhost;Database=Fabrikam;User ID=sqluser;Password=password;".
Publish Profile	(Optional) Publish profile provide fine-grained control over SQL Server database deployments. Specify the path to the Publish profile XML file on the target machine or on a UNC share that is accessible by the machine administrator's credentials.

ARGUMENT	DESCRIPTION
Additional Arguments	(Optional) Additional SqlPackage.exe arguments that will be applied when deploying the SQL Server database like, /p:IgnoreAnsiNulls=True /p:IgnoreComments=True. These arguments will override the settings in the Publish profile XML file (if provided).
Additional Arguments	(Optional) Additional Invoke-Sqlcmd arguments that will be applied when deploying the SQL Server database.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Deploy: SSH

7/10/2018 • 2 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

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).

YAML snippet

```
# SSH
# Run shell commands or a script on a remote machine using SSH
- task: SSH@0
  inputs:
    sshEndpoint:
    #runOptions: 'commands' # Options: commands, script, inline
    #commands: # Required when runOptions == Commands
    #scriptPath: # Required when runOptions == Script
    #inline: # Required when runOptions == Inline
    #args: # Optional
    #failOnStdErr: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
SSH endpoint	The name of an SSH service connection containing connection details for the remote machine. The hostname or IP address of the remote machine, the port number, and the user name are required to create an SSH service connection. - The private key and the passphrase must be specified for authentication. - A password can be used to authenticate to remote Linux machines, but this is not supported for macOS or Windows systems.
Run	Choose to run either shell commands or a shell script on the remote machine.

ARGUMENT	DESCRIPTION
Commands	The shell commands to run on the remote machine. This parameter is available only when **Commands** is selected for the **Run** option. Enter each command together with its arguments on a new line of the multi-line textbox. To run multiple commands together, enter them on the same line separated by semicolons. Example: `cd /home/user/myFolder;build`
Shell script path	Path to the shell script file to run on the remote machine. This parameter is available only when **Shell script** is selected for the **Run** option.
Arguments	The arguments to pass to the shell script. This parameter is available only when **Shell script** is selected for the **Run** option.
Advanced - Fail on STDERR	If this option is selected (the default), the build will fail if the remote commands or script write to **STDERR**.
Control options	See Control options

See also

- Install SSH Key task
- Copy Files Over SSH
- Blog post SSH build task

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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

7/9/2018 • 3 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Copy files to remote machines.

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.

YAML snippet

```
# Windows Machine File Copy
# Copy files to remote machine(s)
- task: WindowsMachineFileCopy@2
  inputs:
    sourcePath:
    #machineNames: # Optional
    #adminUserName: # Optional
    #adminPassword: # Optional
    targetPath:
    #cleanTargetBeforeCopy: false # Optional
    #copyFilesInParallel: true # Optional
    #additionalArguments: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Source	The path to the files to copy. Can be a local physical path such as `c:\files` or a UNC path such as `\\myserver\fileshare\files`. You can use pre-defined system variables such as `$(Build.Repository.LocalPath)` (the working folder on the agent computer), which makes it easy to specify the location of the build artifacts on the computer that hosts the automation agent.
Machines	A comma-separated list of machine FQDNs or IP addresses, optionally including the port number. Can be: - The name of an Azure Resource Group. - A comma-delimited list of machine names. Example: `dbserver.fabrikam.com, dbserver_int.fabrikam.com:5986,192.168.34:5986` - An output variable from a previous task.

ARGUMENT	DESCRIPTION
Admin Login	The username of either a domain or a local administrative account on the target host(s). - Formats such as **domain\username**, **username**, and **machine-name\username** are supported. - UPN formats such as **username@domain.com** and built-in system accounts such as **NT Authority\System** are not supported.
Password	The password for the administrative account specified above. Consider using a secret variable global to the build or release pipeline to hide the password. Example: `$(passwordVariable)`
Destination Folder	The folder on the Windows machine(s) to which the files will be copied. Example: `C:\FabrikamFibre\Web`
Advanced - Clean Target	Set this option to delete all the files in the destination folder before copying the new files to it.
Advanced - Copy Files in Parallel	Set this option to copy files to all the target machines in parallel, which can speed up the copying process.
Advanced - Additional Arguments	Arguments to pass to the RoboCopy process. Example: `/min:33553332 /l`
Select Machines By	Depending on how you want to specify the machines in the group when using the **Filter Criteria** parameter, choose **Machine Names** or **Tags**.
Filter Criteria	Optional. A list of machine names or tag names that identifies the machines that the task will target. The filter criteria can be: - The name of an Azure Resource Group. - An output variable from a previous task. - A comma-delimited list of tag names or machine names. Format when using machine names is a comma-separated list of the machine FDQNs or IP addresses. Specify tag names for a filter as {TagName}**:**{Value} Example: `Role:DB;OS:Win8.1`
Control options	See Control options

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

I get a system error 53 when using this task. Why?

Typically this occurs when the specified path cannot be located. This may be due to a firewall blocking the necessary ports for file and printer sharing, or an invalid path specification. For more details, see Error 53 on Technet.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Tool: Go Tool Installer

7/9/2018 • 2 minutes to read • Edit Online

Finds or downloads specific version of Go tool into tools cache and adds it to the PATH. Use this task to change the version of Go Lang used in subsequent tasks

YAML snippet

```
# Go Tool Installer
# Finds or downloads a specific version of Go in the tools cache and adds it to the PATH. Use this to set the
version of Go used in subsequent tasks.
- task: GoTool@0
  inputs:
    #version: '1.10'
    #goPath: # Optional
    #goBin: # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version	(Required) Go tool version to download and install. Example: 1.9.3
GOPATH	(Optional) Value for the GOPATH environment variable.
GOBIN	(Optional) Value for the GOBIN environment variable.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Tool: Helm tool installer

7/9/2018 • 2 minutes to read • Edit Online

Install Helm and Kubernetes on agent machine.

YAML snippet

```
# Helm tool installer
# Install Helm and Kubernetes on agent machine.
- task: HelmInstaller@0
  inputs:
    #helmVersion: '2.8.2'
    #checkLatestHelmVersion: true # Optional
    #installKubectl: true
    #kubectlVersion: '1.8.9' # Optional
    #checkLatestKubectl: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Helm Version Spec	(Required) Specify the version of Helm to install
Check for latest version of Helm	(Optional) Check for latest version of Helm.
Install Kubectl	(Required) Install Kubectl.
Kubectl Version Spec	(Optional) Specify the version of Kubectl to install
Check for latest version of kubectl	(Optional) Check for latest version of kubectl.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Tool: Java Tool Installer

7/9/2018 • 2 minutes to read • Edit Online

Acquires a specific version of Java from a user supplied Azure blob, a location in the souce 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

YAML snippet

```
# Java Tool Installer
# Acquires a specific version of Java from a user supplied Azure blob or the tools cache and sets JAVA_HOME.
Use this task to change the version of Java used in Java tasks.
- task: JavaToolInstaller@0
  inputs:
    #versionSpec: '1.8'
    jdkArchitectureOption: # Options: x64, x86
    jdkSourceOption: # Options: azureStorage, localDirectory
    #jdkFile: # Required when jdkSourceOption == LocalDirectory
    #azureResourceManagerEndpoint: # Required when jdkSourceOption == AzureStorage
    #azureStorageAccountName: # Required when jdkSourceOption == AzureStorage
    #azureContainerName: # Required when jdkSourceOption == AzureStorage
    #azureCommonVirtualFile: # Required when jdkSourceOption == AzureStorage
    jdkDestinationDirectory:
    #cleanDestinationDirectory: true
```

Arguments

ARGUMENT	DESCRIPTION
JDK Version	Specify which JDK version to download and use.
JDK Architecture	Specify the bit version of the JDK.
JDK source	Specify the source for the compressed JDK, either Azure blob storage or a local directory on the agent or source repository.
JDK file	Applicable when JDK is located in a local directory. Specify the path to the folder that contains the compressed JDK. The path could be in your source repository or a local path on the agent.
Azure Subscription	Applicable when the JDK is located in Azure Blob storage. Specify the Azure Resource Manager subscription for the JDK.
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.

ARGUMENT	DESCRIPTION
Container Name	Applicable when the JDK is located in Azure Blob storage. Specify the name of the container in the storage account in which the JDK is located.
Common Virtual Path	Applicable when the JDK is located in Azure Blob storage. Specify the path to the JDK inside the Azure storage container.
Destination directory	Specify the destination directory into which the JDK should be extracted.
Clean destination directory	Select this option to clean the destination directory before the JDK is extracted into it.
Control options	See Control options.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see Tool installers.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Tool: .NET Core Tool Installer

7/9/2018 • 2 minutes to read • Edit Online

Acquires a specific version of .NET Core from internet or the tools cache and adds it to the PATH. Use this task to change the version of .NET Core used in subsequent tasks.

YAML snippet

```
# .NET Core Tool Installer
# Acquires a specific version of .NET Core from internet or the tools cache and adds it to the PATH. Use this
task to change the version of .NET Core used in subsequent tasks.
- task: DotNetCoreInstaller@0
  inputs:
    #packageType: 'sdk' # Options: runtime, sdk
    #version: '1.0.4'
```

Arguments

ARGUMENT	DESCRIPTION
Package to install	(Required) Please select whether to install only runtime or full SDK.
Version	(Required) Specify exact version of .NET Core SDK or runtime to install. Examples: 1. To install 1.0.4 SDK, use 1.0.4 2. To install 1.1.2 runtime, use 1.1.2 2. To install 2.0 preview 2 runtime, use 2.0.0-preview2-25407-01 For getting more details about exact version, refer here
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Tool: Node Tool Installer

7/9/2018 • 2 minutes to read • Edit Online

Build

Finds or downloads and caches the specified version of Node.js and adds it to the PATH

Demands

None

YAML snippet

```
# Node Tool Installer
# Finds or Downloads and caches specified version spec of Node and adds it to the PATH.
- task: NodeTool@0
  inputs:
    #versionSpec: '6.x'
    #checkLatest: false # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version Spec	Specify which Node.js version you want to use. Examples: `7.x`, `6.x`, `6.10.0`, `>=6.10.0`
Check for Latest Version	Select if you want the agent to check for the latest available version that satisfies the version spec. For example, you select this option because you run this build on your self-hosted agent and you want to always use the latest `6.x` version. TIP If you're using our Microsoft-hosted agents, you should leave this check box cleared. We update the Microsoft-hosted agents on a regular basis, but they're often slightly behind the latest version. So selecting this box will result in your build spending a lot of time updating to a newer minor version.
Control options	See Control options.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see Tool installers.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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.

Tool: Use Python Version

7/11/2018 • 2 minutes to read • Edit Online

Selects a version of Python to run on an agent. Optionally adds it to PATH.

Demands

None

Prerequisites

- A Microsoft-hosted agent, or a self-hosted agent with its tool cache configured (see Q&A).

This task will fail if no Python versions are found in the tool cache. Available Python versions on Microsoft-hosted agents can be found here.

YAML snippet

```
# Use Python Version
# Retrieves the specified version of Python from the tool cache. Optionally add it to PATH.
- task: UsePythonVersion@0
  inputs:
    #versionSpec: '>= 3.3'
    #addToPath: true
    #architecture: 'x64' # Options: x86, x64
```

Arguments

ARGUMENT	DESCRIPTION
Version spec	Version range or exact version of a Python version to use.
Add to PATH	Whether to prepend the retrieved Python version to the PATH environment variable to make it available in subsequent tasks or scripts without using the output variable.
Advanced - Architecture	The target architecture (x86, x64) of the Python interpreter.

If the task completes successfully, the task's output variable will contain the directory of the Python installation:

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see Tool installers.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 can I configure a self-hosted agent to use this task?

You can run this task on a self-hosted agent with your own Python versions. To run this task on a self-hosted agent, set up your agent's tool cache by following the instructions here. The tool name to use is "Python."

Tool: Use Ruby Version

7/11/2018 • 2 minutes to read • Edit Online

Selects a version of Ruby to run on an agent. Optionally adds it to PATH.

Demands

None

Prerequisites

- A Microsoft-hosted agent, or a self-hosted agent with its tool cache configured (see Q&A).

This task will fail if no Ruby versions are found in the tool cache. Available Ruby versions on Microsoft-hosted agents can be found here.

YAML snippet

```
# Use Ruby Version
# Retrieves the specified version of Ruby from the tool cache. Optionally add it to PATH.
- task: UseRubyVersion@0
  inputs:
    #versionSpec: '>= 2.4'
    #addToPath: true # Optional
```

Arguments

ARGUMENT	DESCRIPTION
Version spec	Version range or exact version of a Ruby version to use.
Add to PATH	Whether to prepend the retrieved Ruby version to the PATH environment variable to make it available in subsequent tasks or scripts without using the output variable.

If the task completes successfully, the task's output variable will contain the directory of the Ruby installation.

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

Where can I learn more about tool installers?

For an explanation of tool installers and examples, see Tool installers.

Do I need an agent?

You need at least one agent to run your build or release. Get an agent.

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 can I configure a self-hosted agent to use this task?

You can run this task on a self-hosted agent with your own Ruby versions. To run this task on a self-hosted agent, set up your agent's tool cache by following the instructions here. The tool name to use is "Ruby."

Tool: Visual Studio Test Platform Installer

7/9/2018 • 2 minutes to read • Edit Online

Acquires the test platform from nuget.org or the tools cache. Satisfies the 'vstest' demand and can be used for running tests and collecting diagnostic data using the Visual Studio Test task.

YAML snippet

```
# Visual Studio Test Platform Installer
# Acquires the test platform from nuget.org or the tools cache. Satisfies the 'vstest' demand and can be used
for running tests and collecting diagnostic data using the Visual Studio Test task.
- task: VisualStudioTestPlatformInstaller@1
  inputs:
    #versionSelector: 'latestPreRelease' # Options: latestPreRelease, specificVersion
    #testPlatformVersion: # Required when versionSelector == SpecificVersion
```

Arguments

ARGUMENT	DESCRIPTION
Version	(Required) Pick whether to install the latest version or a specific version of the Visual Studio Test Platform.
Test Platform Version	(Required) Specify the version of Visual Studio Test Platform to install on the agent. Available versions can be viewed on nuget.
CONTROL OPTIONS	

Open source

This task is open source on GitHub. Feedback and contributions are welcome.

Q & A

File matching patterns reference

5/29/2018 • 2 minutes 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.tiff
image.png
image.ico
```

The pattern would match:

```
image.png
image.ico
```

Character set examples

Example 1: Given the pattern `Sample[AC].dat` and files:

```
SampleA.dat
SampleB.dat
SampleC.dat
SampleD.dat
```

The pattern would match:

```
SampleA.dat
SampleC.dat
```

Example 2: Given the pattern `Sample[A-C].dat` and files:

```
SampleA.dat
SampleB.dat
SampleC.dat
SampleD.dat
```

The pattern would match:

```
SampleA.dat
SampleB.dat
SampleC.dat
```

Example 3: Given the pattern `Sample[A-CEG].dat` and files:

```
SampleA.dat
SampleB.dat
SampleC.dat
SampleD.dat
SampleE.dat
SampleF.dat
SampleG.dat
SampleH.dat
```

The pattern would match:

```
SampleA.dat
SampleB.dat
SampleC.dat
SampleE.dat
SampleG.dat
```

Exclude pattern examples

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

7/9/2018 • 7 minutes to read • Edit Online

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

Some tasks, such as the Azure App Service Deploy task version 3 and later and the IIS Web App Deploy task, allow users to configure the package based on the environment specified. These tasks use **msdeploy.exe**, which supports the overriding of values in the **web.config** file with values from the **parameters.xml** file.

Configuration substitution is specified in the **File Transform and Variable Substitution Options** section of the settings for the tasks. The transformation and substitution options are:

- XML transformation
- XML variable substitution
- JSON variable substitution

When the task runs, it first performs XML transformation, XML variable substitution, and JSON variable substitution on configuration and parameters files. Next, it invokes **msdeploy.exe**, which uses the **parameters.xml** file to substitute values in the **web.config** file.

XML Transformation

XML transformation supports transforming the configuration files (`*.config` files) by following Web.config Transformation Syntax and is based on the environment to which the web package will be deployed. This option is useful when you want to add, remove or modify configurations for different environments. Transformation will be applied for other configuration files including Console or Windows service application configuration files (for example, **FabrikamService.exe.config**).

Configuration transform file naming conventions

XML transformation will be run on the `*.config` file for transformation configuration files named `*.Release.config` or `*.<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 `ConnectionStrings` element.
- It modifies value of `Webpages:Enabled` inside the `appSettings` element.
- It removes the `debug` attribute from the `compilation` element inside the `System.Web` element.

> For more information, see Web.config Transformation Syntax for Web Project Deployment Using Visual Studio

2. Create a release pipeline with an environment named **Release**.

3. Add an **Azure App Service Deploy** task and set (tick) the **XML transformation** option.

4. Save the release pipeline and start a new release.

5. Open the `Web.config` file to see the transformations from `Web.Release.config`.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="DefaultConnection"
         connectionString="Data Source=(LocalDb)\\MSDB;DbFilename=aspcore-local.mdf;" />
  <add name="MyDB"
       connectionString="Data Source=ReleaseSQLServer;Initial Catalog=MyReleaseDB;Integrated
Security=True" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="true" />
  </appSettings>
  <system.web>
    <authentication mode="None" />
    <compilation targetFramework="4.5" />
  </system.web>
</configuration>
```

Note:

- You can use this technique to create a default package and deploy it to multiple 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 (`*.config` files) inside web packages and XML parameters files (`parameters.xml`). In this way, the same package can be configured based on the environment to which it will be deployed.

Variable substitution takes effect only on the `applicationSettings`, `appSettings`, `connectionStrings`, and `configSections` elements of configuration files.

XML variable substitution example

As an example, consider the task of changing the following values in `Web.config`:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <configSection>
        <section name="entityFramework" />
    </configSection>
    <connectionStrings>
        <!-- Change connectionString in this line: -->
        <add name="DefaultConnection"
             connectionString="Data Source=(LocalDB)\LocalDB;FileName=Local.mdf" />
    </connectionStrings>
    <appSettings>
        <add key="ClientValidationEnabled" value="true" />
        <add key="UnobstructiveJavascriptEnabled" value="true" />
        <!-- Change AdminUserName in this line: -->
        <add key="AdminUserName" value="__AdminUserName__" />
        <!-- Change AdminPassword in this line: -->
        <add key="AdminPassword" value="__AdminPasword__" />
    </appSettings>
    <entityFramework>
        <defaultConnectionFactory type="System.Data.Entity.LocalDbConnectionFactory">
            <parameters></parameters>
        </defaultConnectionFactory>
        <providers>
            <!-- Change invariantName in this line: -->
            <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer" />
        </providers>
    </entityFramework>
</configuration>
```

1. Create a release pipeline with an environment named **Release**.

2. Add an **Azure App Service Deploy** task and set (tick) the **XML variable substitution** option.

3. Define the required values in release pipeline variables:

NAME	VALUE	SECURE	SCOPE
DefaultConnection	Data Source=(ProdDB)\MSSQLProdDB;AttachFileName=Local.mdf	No	Release
AdminUserName	ProdAdminName	No	Release
AdminPassword	[your-password]	Yes	Release
invariantName	System.Data.SqlClientExtension	No	Release

4. Save the release pipeline and start a new release.

5. Open the `Web.config` file to see the variable substitutions.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <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="UnobstructiveJavascriptEnabled" value="true" />
        <add key="AdminUserName" value="ProdAdminName" />
        <add key="AdminPassword" value="*password_masked_for_display*" />
    </appSettings>
    <entityFramework>
        <defaultConnectionFactory type="System.Data.Entity.LocalDbConnectionFactory">
            <parameters></parameters>
        </defaultConnectionFactory>
        <providers>
            <provider invariantName="System.Data.SqlClientExtension"
                      type="System.Data.Entity.SqlServer" />
        </providers>
    </entityFramework>
</configuration>
```

Note:

- By default, ASP.NET applications have a default parameterized connection attribute. These values are overridden only in the `parameters.xml` file inside the web package.
- Because substitution occurs before deployment, the user can override the values in `Web.config` using `parameters.xml` (inside the web package) or a `setparameters` file.

JSON variable substitution

This feature substitutes values in the JSON configuration files. It overrides the values in the specified JSON configuration files (for example, `appsettings.json`) with the values matching names of release pipeline and 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": {
      "Admininstrators": ["Admin-1", "Admin-2"],
      "Users": ["Vendor-1", "vendor-3"]
    },
    "FeatureFlags": {
      "Preview": [
        {
          "newUI": "AllAccounts"
        },
        {
          "NewWelcomeMessage": "Newusers"
        }
      ]
    }
  }
}
```

The task is to override the values of **ConnectionString**, **DebugMode**, the first of the **Users** values, and **NewWelcomeMessage** at the respective places within the JSON file hierarchy.

1. Create a release pipeline with 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 pipeline 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 pipeline and start a new release.

5. After the transformation, the JSON will contain the following:

```
{
  "Data": {
    "DefaultConnection": {
      "ConnectionString": "Data Source=(prodDB)\MSDB;AttachDbFilename=prod.mdf;"
    },
    "DebugMode": "disabled",
    "DBAccess": {
      "Admininstrators": ["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)

7/17/2018 • 9 minutes to read • Edit Online

VSTS | TFS 2018 | TFS 2017 | TFS 2015

> **NOTE**
> Build and release *pipelines* are called *definitions* in TFS 2018 and in older versions. *Service connections* are called *service endpoints* in TFS 2018 and in older versions.

To support security of your build and release operations, you can add users to a built-in security group, set individual permissions for a user or group, or add users to pre-defined roles. You manage security for the following objects from 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 connection. Membership of these roles can be configured hierarchically, as well as at either project level or individual entity level.

Build permissions

Permissions in Build follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual build pipeline.

To set the permissions at project level for all build pipelines in a project, choose **Security** from the action bar on the main page of Builds hub.

To set or override the permissions for a specific build pipeline, choose **Security** from the context menu of the build pipeline.

The following permissions are defined in Build. All of these can be set at both the levels.

PERMISSION	DESCRIPTION
Administer build permissions	Can change any of the other permissions listed here.
Queue builds	Can queue new builds.
Delete build pipeline	Can delete build pipeline(s).
Delete builds	Can delete builds for a pipeline. Builds that are deleted are retained in the **Deleted** tab for a period of time before they are destroyed.
Destroy builds	Can delete builds from the **Deleted** tab.
Edit build pipeline	Can save any changes to a build pipeline, including configuration variables, triggers, repositories, and retention policy.
Edit build quality	Can add tags to a build.
Override check-in validation by build	Applies to TFVC gated check-in builds. This does not apply to PR builds.
Retain indefinitely	Can toggle the retain indefinitely flag on a build.
Stop builds	Can stop builds queued by other team members or by the system.
View build pipeline	Can view build pipeline(s).
View builds	Can view builds belonging to build pipeline(s).
Update build information	It is recommended to leave this alone. It's intended to enable service accounts, not team members.
Manage build qualities	*Only applies to XAML builds*
Manage build queue	*Only applies to XAML builds*

Default values for all of these permissions are set for team project collections and project groups. For example, **Project Collection Administrators**, **Project Administrators**, and **Release Administrators** are given all of the above permissions by default. **Contributors** are given all permissions except **Administer release permissions**. **Readers**, by default, are denied all permissions except **View release pipeline** and **View releases**.

When it comes to security, there are different best practices and levels of permissiveness. While there's no one right

way to handle permissions, we hope these examples help you empower your team to work securely with builds.

- By default, contributors in a project cannot create or edit build pipelines. To grant permissions to work on build pipelines, select *Contributors* and set the **Edit build pipeline** permission to *Allow*.
- In many cases you probably also want to set **Delete build pipeline** to *Allow*. Otherwise these team members can't delete even their own build pipelines.
- Without **Delete builds** permission, users cannot delete even their own completed builds. However, keep in mind that they can automatically delete old unneeded builds using retention policies.
- We recommend that you do not grant these permissions directly to a person. A better practice is to add the person to the build administrator group or another group, and manage permissions on that group.

Release permissions

Permissions in Release Management follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual release pipeline. Some of the permissions can also be overridden on a specific environment within a pipeline. 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 pipelines** and choose **Security**.

To set or override the permissions for a specific release pipeline, open the shortcut menu from the ▾ icon next to that pipeline name. Then choose **Security** to open the **Permissions** dialog.

To specify security settings for individual environments in a release pipeline, open the **Permissions** dialog by choosing **Security** on the shortcut menu that opens from the ellipses (**...**) on an environment in the release pipeline editor.

The following permissions are defined in Release Management. The scope column explains whether the permission can be set at the project, release pipeline, or environment level.

PERMISSION	DESCRIPTION	SCOPES
Administer release permissions	Can change any of the other permissions listed here.	Project, Release pipeline, Environment
Create releases	Can create new releases.	Project, Release pipeline
Delete release pipeline	Can delete release pipeline(s).	Project, Release pipeline
Delete release environment	Can delete environment(s) in release pipeline(s).	Project, Release pipeline, Environment
Delete releases	Can delete releases for a pipeline.	Project, Release pipeline
Edit release pipeline	Can save any changes to a release pipeline, including configuration variables, triggers, artifacts, and retention policy as well as configuration within an environment of the release pipeline. To make changes to a specific environment in a release pipeline, the user also needs **Edit release environment** permission.	Project, Release pipeline

PERMISSION	DESCRIPTION	SCOPES
Edit release environment	Can edit environment(s) in release pipeline(s). To save the changes to the release pipeline, the user also needs **Edit release pipeline** permission. This permission also controls whether a user can edit the configuration inside the environment of a specific release instance. The user also needs **Manage releases** permission to save the modified release.	Project, Release pipeline, Environment
Manage deployments	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 **Deploy** or **Redeploy** 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 pipeline, Environment
Manage release approvers	Can add or edit approvers for environment(s) in release pipeline(s). This permissions also controls whether a user can edit the approvers inside the environment of a specific release instance.	Project, Release pipeline, Environment
Manage releases	Can edit the configuration in releases. To edit the configuration of a specific environment in a release instance, the user also needs **Edit release environment** permission.	Project, Release pipeline
View release pipeline	Can view release pipeline(s).	Project, Release pipeline
View releases	Can view releases belonging to release pipeline(s).	Project, Release pipeline

Default values for all of these permissions are set for team project collections and project groups. For example, **Project Collection Administrators**, **Project Administrators**, and **Release Administrators** are given all of the above permissions by default. **Contributors** are given all permissions except **Administer release permissions**. **Readers**, by default, are denied all permissions except **View release pipeline** and **View releases**.

Task group permissions

Task group permissions follow a hierarchical model. Defaults for all the permissions can be set at the project level and can be overridden on an individual task group pipeline.

You use task groups to encapsulate a sequence of tasks already defined in a build or a release pipeline into a single reusable task. You define and manage task groups in the **Task groups** tab of the **Build and Release** hub.

PERMISSION	DESCRIPTION
Administer task group permissions	Can add and remove users or groups to task group security.
Delete task group	Can delete a task group.
Edit task group	Can create, modify, or delete a task group.

Library roles and permissions

Permissions for library artifacts, such as variable groups and secure files, are managed by roles. You use a variable group to store values that you want to make available across multiple build and release pipelines. You define and manage variable groups and secure files in the **Library** tab of the **Build and Release** hub.

ROLE	DESCRIPTION
Administrator	Can use and manage library items.
Reader	Can only read library items.
User	Can use library items, but not manage them.

Service connection security roles

You add users to the following roles from the project-level admin context, **Services** page. To create and manage these resources, see Service connections for build and release.

ROLE	DESCRIPTION
User	Can use the endpoint when authoring build or release definitions.
Administrator	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
Reader	Can only view deployment pools.
Service Account	Can view agents, create sessions, and listen for jobs from the agent pool.
User	Can view and use the deployment pool for creating deployment groups.

ROLE	DESCRIPTION
Administrator	Can administer, manage, view and use deployment pools.

Related notes

- Set build and release permissions
- Default permissions and access
- Permissions and groups reference