Azure DevOps documentation

Collaborate on software development through source control, work tracking, and continuous integration and delivery, both on-premises and in the cloud!

OVERVIEW
What is Azure DevOps?

OVERVIEW
Azure DevOps Services vs. Azure DevOps Server

WHAT'S NEW
What's new in Azure DevOps?

WHAT'S NEW
What's new in Azure DevOps documentation?

GET STARTED
Get started for free

GET STARTED
Create your first pipeline

HOW-TO GUIDE
Connect to a project

HOW-TO GUIDE
Authenticate with Azure DevOps

Azure Boards

- What is Azure Boards?
- About default processes
- Create your backlog
- Plan & track work
- Add or modify work items in bulk
- See more >

Azure Repos

- What is Azure Repos?
- Clone an existing Git repository
- Get started with Git & Visual Studio
- Connect to your Git repos with SSH

Azure Pipelines

- What is Azure Pipelines?
- Create your first pipeline
- YAML schema
- Define variables
- Manage service connections
- See more >

Review and merge code with pull requests

See more >

Azure Test Plans

What is Azure Test Plans?

Create a test plan

Run automated tests

Create manual tests

See more >

Azure Artifacts

What is Azure Artifacts?

Artifacts feeds

Upstream sources

Configure permissions

Artifacts in Azure Pipelines

See more >

Azure DevOps Server

Administrative tasks

Setup and upgrade requirements

Install

Migrate

Build & deploy your apps

ASP.NET

.NET Core

C/C++ with GCC

C/C++ with VC++

Docker

Java

JavaScript & Node.js

Python

Azure Web Apps

Windows VM

Kubernetes

Azure SQL Database

Configure & customize resources

Settings & usage

About settings
Manage your project
User preferences
Teams
Projects
Public projects
Organizations
Pipeline agent pools & queues
Auditing
See more >

Security, permissions, & access

About permissions, access, and security groups
About access levels
Billing overview
Add or remove users or groups
Change project-level permissions
Change project collection-level permissions
See more >

Cross-service, Azure, & GitHub integration

Cross-service integration overview
End-to-end traceability
Azure DevOps and GitHub integration
Deploy to Azure
Work across projects
Azure Boards & GitHub
See more >

Azure Boards customization

Configure & customize Azure Boards
Compare processes
Area & iteration paths
Customize Inherited process
On-premises XML process model
Hosted XML process model
Process templates
See more >

Other services

Analytics & reporting

Dashboards, charts, reports, & widgets
Widget catalog
Work tracking query-based charts

Wiki, search, & navigation

Create a wiki for your project
Syntax guidance for Markdown usage in Wiki
Search

Azure DevOps and Power BI

What are Analytic views?

See more >

Web portal navigation

See more >

Marketplace & Extensibility

Authentication overview

Discover extensions

Develop extensions

Service hooks

See more >

Developer resources

Azure DevOps CLI

Azure DevOps CLI quick reference

Git command reference

Team Foundation version control commands

Azure DevOps Services Demo Generator

REST APIs

DevOps resource center

See more >

Blogs - Twitter - Stack Overflow - Issue Reporting - Developer Community - Service Status

Get started with Azure DevOps documentation

Start using Azure DevOps to collaborate on code, build and deploy apps, or plan and track work.

About Azure DevOps

OVERVIEW

What is Azure DevOps?

What is Azure Boards?

What is Azure Repos?

What is Azure Pipelines?

What is Azure Test Plans?

Get started as a Stakeholder

Sign up for free

QUICKSTART

Azure DevOps (includes all services)

Azure Boards

Azure Repos

Azure Pipelines

Azure Artifacts

HOW-TO GUIDE

Connect to a project

Manage your project

QUICKSTART

Manage your project

Add users to a team or project

Add a project administrator

Add a project collection administrator

Create a new project

Delete or restore a project

Set up billing for your organization

CONCEPT

Default permissions and access

Permission lookup guide

About settings

Security best practices

Plan and track work

QUICKSTART

Plan and track work

Create your backlog

Define sprints

Use boards

Collaborate on code

QUICKSTART

Clone a repository

Create a repository

Authenticate with SSH

Search code across projects

Set Git repo permissions

Build and deploy your apps

QUICKSTART

Create your first pipeline

Build multiple branches

REFERENCE

YAML schema

Pipeline variables

Set your preferences

QUICKSTART

Change profile preferences

Manage your notifications

Preview new features

Set favorites

Navigate

HOW-TO GUIDE

Web portal navigation

Team Explorer navigation

Use the command line

Migrate

CONCEPT

Migration overview

Get started

Prerequisites

Validate

Prepare test run

Test run

Migrate

Post-migration

Troubleshoot

Azure Artifacts documentation

Code once and share packages across your organization. Host your private NuGet, npm, Maven, Python, and Universal Packages with Azure Artifacts for more reliable and scalable builds.

Get started

CONCEPT

Artifacts overview

QUICKSTART

Best practices

Artifacts storage consumption

Package sizes and count limits

Configure Feeds

CONCEPT

What are feeds?

What are feed views?

QUICKSTART

Configure permissions

Upstream sources overview

Configure upstream sources

Artifacts in Azure Pipelines

CONCEPT

Overview

QUICKSTART

Pipeline Artifacts

Release Artifacts and Artifact sources

Publish and download Artifacts

Pipeline caching

Symbols

CONCEPT

What are symbol files?

HOW-TO GUIDE

Debug with Visual Studio

Debug with WinDbg

Publish symbols with Azure Pipelines

NuGet

QUICKSTART

Get started

HOW-TO GUIDE

Publish NuGet packages (CLI)

Publish NuGet packages with Azure Pipelines (YAML/Classic)

Install NuGet packages with Visual Studio

NuGet.org upstream source

npm

QUICKSTART

Get started

HOW-TO GUIDE

Publish npm packages (CLI)

Publish npm packages with Azure Pipelines (YAML/Classic)

Set up your project and connect to Azure Artifacts

Use npm scopes

Npmjs.com upstream source

Maven

QUICKSTART

Get started

HOW-TO GUIDE

Project setup

Install Maven Artifacts

Use packages from Maven Central

Google Maven Repository upstream source

Gradle Plugins upstream source

JitPack upstream source

Python

QUICKSTART

Get started

HOW-TO GUIDE

Publish and download Python packages (CLI)

Publish and download Python packages with Azure Pipelines (YAML/Classic)

Universal Packages

QUICKSTART

Get started

HOW-TO GUIDE

Publish and download Universal Packages with Azure Pipelines (YAML/Classic)

Universal Packages upstream sources

Azure Boards documentation

Plan, track, and discuss work across teams. Define and update issues, bugs, user stories, & other work with customizable Scrum, Kanban, and Agile tools.

About Azure Boards

OVERVIEW

What is Azure Boards?

About work items

CONCEPT

About default processes

Agile process workflow

Best practices for Agile project management

Get started

QUICKSTART

Sign up for Azure Boards

Plan & track work

Get started as a Stakeholder

Plan your project

OVERVIEW

About Backlogs & Agile project management

QUICKSTART

Create your backlog

Define features & epics

Organize your backlog

TUTORIAL

Bulk add or modify work items (Excel)

Bulk add or modify work items (Web)

Implement Kanban

QUICKSTART

Board quickstart

TUTORIAL

Board overview

Add columns

Customize your board

Implement Scrum

OVERVIEW

About Sprints & Scrum

TUTORIAL

Assign work to sprints

Set capacity

Monitor sprint burndown

List & manage work items

TUTORIAL

Create managed queries

Do ad hoc searches

Define, triage, & manage bugs

Remove, delete, or restore work items

Collaborate

QUICKSTART

Connect Azure Boards to GitHub

Link GitHub commits, pull requests, & branches to work items

CONCEPT

Link work items to other objects

TUTORIAL

Use work item templates

Add tags to work items

Azure Boards with Microsoft Teams

Configure & customize

CONCEPT

About teams & Agile tools

Configure & customize Boards

TUTORIAL

Set area paths

Set iteration (sprint) paths

Add a custom field

Customize the workflow

See more...

Agile at scale

TUTORIAL

Portfolio management

Review team deliverables & plans

Implement Scaled Agile Framework®

Troubleshoot

HOW-TO GUIDE

Fix reordering & nesting issues

Set up your Backlogs & Boards

FAQs

Quick reference

REFERENCE

Work items quick reference

Query quick reference

Work item field index

Developer resources

REFERENCE

Azure DevOps CLI

REST API

Azure Pipelines documentation

Implement continuous integration and continuous delivery (CI/CD) for the app and platform of your choice.

Get started

CONCEPT

What is Azure Pipelines?

Use Azure Pipelines

Key concepts for user new to Azure Pipelines

Classic release pipelines

QUICKSTART

Sign up for free

Create your first pipeline

Clone or import a pipeline

Customize your pipeline

Build, deploy, test-any language, any ecosystem

TUTORIAL

Container image

.NET Core apps

Anaconda

Android

Azure Kubernetes Service

Java

JavaScript & Node.js apps

Python apps

PHP

See more >

Key concepts

CONCEPT

Agents

Conditions

Expressions

Environments

Jobs

Runtime parameters

Stages

Tasks

Templates

Triggers

Reference guidance

REFERENCE

YAML schema

Predefined variables

Release & artifacts variables

Variable groups

Task index

Build any repository

HOW-TO GUIDE

GitHub

Azure Repos Git

Team Foundation version control

Bitbucket Cloud

Subversion

Generic Git

TUTORIAL

Build multiple Git branches

Deploy to Azure

OVERVIEW

Deploy to Azure

TUTORIAL

Connect to Azure

Deploy to Azure SQL Database

Deploy to Azure Kubernetes Service

Use secrets from Azure Key Vault

Manage agents & self-hosted agents

HOW-TO GUIDE

Deploy on Windows

Microsoft-hosted agents

Self-hosted Linux agents

Self-hosted macOS agents

Set up continuous testing

TUTORIAL

Configure for UI testing

UI testing using Selenium

Review code coverage results

Review test results

Manage flaky tests

Analyze test results

Configure resources

HOW-TO GUIDE

Create and manage agent pools

Create and manage deployment groups

Create and manage service connections

Set build and release retention policies

Troubleshoot

HOW-TO GUIDE

Troubleshoot builds and releases

Run a release in debug mode

Troubleshoot Azure Resource Manager service connections

Publish packages

TUTORIAL

Publish & download artifacts

Publish NuGet packages

Publish npm packages

Publish Python packages

Publish and download Universal Packages

Publish Maven Packages

Publish symbols for debugging

Integrate with other apps

HOW-TO GUIDE

Integrate with Slack

Integrate with Microsoft Teams

Integrate with service hooks

Azure Repos documentation

Collaborate on code development using free Git public and private repositories, pull requests, and code review.

About Azure Repos

VIDEO

Use Pull Requests with Azure DevOps

OVERVIEW

What is Azure Repos?

Compare Git and TFVC

Connect & authentication

OVERVIEW

Authentication overview

Connect to an Azure DevOps project

HOW-TO GUIDE

Use SSH key authentication

Use Git Credential Manager

Get started with Git or GitHub

GET STARTED

Commit, share, and sync your code

Clone a repository

Import a Git repo into your project

Update code with fetch and pull

Undo changes in your Git repo

TUTORIAL

Save changes with Git commits

Sync changes to a remote Git repo

Resolve Git merge conflicts

Manage Git repos & branches

HOW-TO GUIDE

Create & manage Git repositories

Set Git repository permissions

Create a new branch

Improve code quality with branch policies

Manage branches

Change the default branch

Ignore files

Set Git branch security and permissions

REFERENCE

Default permissions

Git command reference

Get started with TFVC

QUICKSTART

Share your code using Visual Studio

HOW-TO GUIDE

Suspend work and manage shelvesets

Use Source Control Explorer to manage files

Get code reviewed

Create work spaces

Branch folders and files

REFERENCE

TFVC command reference

Configure TFVC resources

HOW-TO GUIDE

Manage TFVC repository permissions

Add TFVC check-in policies

Developer resources

REFERENCE

Azure DevOps CLI

Git REST API

TFVC REST API

Azure Test Plans documentation

Improve your overall code quality by using manual and exploratory testing services for your apps.

About Azure Test Plans

OVERVIEW

What is Azure Test Plans?

Navigate Test Plans

Store custom data in test plan and/or test result

Run manual tests

QUICKSTART

Create test plans and suites

Create manual test cases

Run manual tests

TUTORIAL

Test different configurations

Test & Feedback extension

QUICKSTART

Install the Test & Feedback extension

Test in Connected mode

Configure Test resources

HOW-TO GUIDE

Set test retention policies

Run automated tests

CONCEPT

Run automated tests from test hub

Associate automated tests with test cases

Set up continuous testing

TUTORIAL

.NET Core apps

Go

Python

UI testing using Selenium

Review code coverage results

Review test results

Analyze test results

Analytics & Reporting documentation

Reporting in Azure DevOps enables software development teams to make data driven decisions and deliver value to their customers faster.

Configure charts & dashboards

OVERVIEW

About dashboards, charts, reports & widgets

Quick reference

Widget catalog

TUTORIAL

Add, rename, and delete dashboards

Add a widget to a dashboard

Configure Burndown & Burnup widgets

View and configure team velocity

HOW-TO GUIDE

Create status and trend query-based charts

Work tracking status and trends

TUTORIAL

Burnup and burndown

Sprint burndown

Cumulative flow

Cycle and Lead time

Velocity

Work item query-based query charts

Pipeline reports

CONCEPT

Review code coverage

Pipeline Analytics reports

Historical graph for agent pools

Explore Analytics OData model

TUTORIAL

Construct OData queries

Define basic queries

OVERVIEW

Best practices to use when querying Analytics

Analytics OData Reference

REFERENCE

Calendar date, Project, and User metadata

Azure Boards metadata

Azure Pipelines metadata

Azure Test Plans metadata

Test status and trends

CONCEPT

Test result trends

Test status, progress, and result charts

Trace test requirements

Test Analytics and failure analysis

Create Power BI reports

OVERVIEW

About Power BI integration

What is Analytics

What are Analytics views?

Reporting roadmap

TUTORIAL

Create an Analytics view

Connect to Power BI data connector

Connect with data by using Power BI and OData queries

Create a Power BI report with an OData Query

Sample Power BI reports

OVERVIEW

Sample reports using OData queries

Data available from Analytics

HOW-TO GUIDE

Add a column for last refresh date

Calculate time-in-state

Extend Analytics with OData

OVERVIEW

About Power BI integration

Quick reference

QUICKSTART

Query work tracking data

Project and organization-scoped queries

Aggregate work tracking data

REFERENCE

Query guidelines for Analytics with OData

View SQL Server Reports (Legacy)

OVERVIEW

Reporting Services reports

HOW-TO GUIDE

Create Excel reports from a work item query

Create a Detailed Report using Report Designer

Add reports to a team project

Settings & Usage documentation

Configure resources and manage settings for an organization, project, team, or user.

Get started

OVERVIEW

About settings

Get started as an administrator

QUICKSTART

Set user preferences

HOW-TO GUIDE

Enable preview features

Manage your organization (cloud)

OVERVIEW

About managing your organization

Plan your organizational structure

CONCEPT

Manage access with Microsoft Entra ID

QUICKSTART

Create an organization

HOW-TO GUIDE

Add users to your organization

Connect your organization to Microsoft Entra ID

Add & manage projects

CONCEPT

About projects & scaling up

Customize your project

OVERVIEW

Manage your project

HOW-TO GUIDE

Create a project

Connect to GitHub

Add & manage teams

CONCEPT

About teams & Agile tools

HOW-TO GUIDE

Add a team

Configure team tools

Define team area paths

Configure team iterations

Add a team administrator

Set alerts or notifications

CONCEPT

About notifications

HOW-TO GUIDE

Set personal notifications

Set team notifications

Audit & usage (cloud)

CONCEPT

Rate limits & usage

QUICKSTART

Access, export, and filter audit logs

Configure Pipelines resources

HOW-TO GUIDE

Create & manage agent pools

Create & manage deployment groups

Set build & release retention policies

Create & manage service connections

Configure Repos resources

OVERVIEW

About branches and branch policies

HOW-TO GUIDE

Create & manage Git repositories

Set Git repository settings and policies

Manage Git branch policies

Add TFVC check-in policies

Configure Test resources

HOW-TO GUIDE

Set test retention policies

Cross-service, Azure, and GitHub integration documentation

Work across Azure DevOps Services and projects, integrate with Azure, GitHub, and other resources, and create and manage public projects.

Work across services

OVERVIEW

Integration overview

Azure DevOps and GitHub integration overview

Deploy to Azure

Integrate with Microsoft Teams

HOW-TO GUIDE

Azure Boards with Microsoft Teams

Azure Repos with Microsoft Teams

Azure Pipelines with Microsoft Teams

Integrate with GitHub

QUICKSTART

Azure Boards & GitHub

Azure DevOps Projects & GitHub

Git experience in Visual Studio

Integrate with Service hooks

OVERVIEW

Integrate with service hooks

TUTORIAL

Create a service hook with WebHooks

Create a service hook with Microsoft Teams

Work across projects

GET STARTED

View work assigned to you

REFERENCE

Work across projects FAQs

Integrate with Slack

HOW-TO GUIDE

Azure Boards with Slack

Azure Repos with Slack

Azure Pipelines with Slack

Track work, code, builds, and releases

OVERVIEW

End-to-end traceability

TUTORIAL

Drive Git development

HOW-TO GUIDE

Link work items to other objects

Link objects to work items

Configure pipelines to support work tracking

Create and manage a public project

OVERVIEW

What is a public project?

QUICKSTART

Make a private project public

Azure Boards Configuration & Customization documentation

Configure and customize Azure Boards including area paths, iteration paths, projects, and processes.

Configure teams

CONCEPT

About teams & Agile tools

QUICKSTART

Configure team tools

Set team area paths

Set team iterations

Configure projects

CONCEPT

Configure & customize Boards

About projects & scaling up

QUICKSTART

Define area paths

Define iteration paths (sprints)

Customize an Inheritance process

OVERVIEW

Inheritance process model

QUICKSTART

Add a custom field

Add a custom work item type

TUTORIAL

Customize a workflow

Apply rules to workflow

Customize a project

Create & manage a process

Customize an On-premises XML process

OVERVIEW

On-premises XML process model

HOW-TO GUIDE

Add or modify a field

Add or modify a work item type

REFERENCE

XML element reference

Customize a process template (on-premises)

OVERVIEW

About processes & process templates

Process template files

HOW-TO GUIDE

Customize a process template

Upload/download process templates

REFERENCE

XML element reference

Reference

REFERENCE

Naming restrictions

Workflow states & categories

Work tracking object limits

Wikis, search, & navigation documentation

Share information with wikis, search within and across projects, and learn how to navigate within Azure DevOps.

Get started

OVERVIEW

Navigate the web portal

Get started with Search

HOW-TO GUIDE

Set your preferences

Enable or disable preview features

Set personal or team favorites

Connect to a project

Add and manage wikis

OVERVIEW

About wikis, READMEs, and Markdown

Provisioned vs published wikis

QUICKSTART

Create a wiki for your project

Add and edit wiki pages

Publish a Git repository to a wiki

Restore a deleted wiki

REFERENCE

Syntax guidance for basic Markdown usage

Syntax guidance for Markdown usage in Wiki

Search

OVERVIEW

Search across all your code, wiki, packages and work items

HOW-TO GUIDE

Search code

Search packages

Search work items

Navigating the web portal

OVERVIEW

Navigation overview

HOW-TO GUIDE

Enable or disable preview features

Work with favorites

Filter lists, boards, and directories

Keyboard shortcuts

Request, get, & track feedback

OVERVIEW

About requesting and providing feedback

HOW-TO GUIDE

Request feedback, Test & Feedback extension

Request feedback, Microsoft Feedback Client

Track feedback

Marketplace & Extensibility documentation

Discover, manage, develop extensions and widgets for integration with Azure DevOps.

Discover & manage Marketplace extensions

CONCEPT

About Azure DevOps Marketplace

HOW-TO GUIDE

Request extensions

Install extensions

Manage extension permissions

Develop extensions

QUICKSTART

Develop a web extension

Extension samples

Extensibility points

TUTORIAL

Create a custom pipelines task

Integrate applications

OVERVIEW

Integration overview

HOW-TO GUIDE

Authorize access to REST APIs with OAuth 2.0

Authorize access with service principals and managed identities

Authenticate access with personal access tokens

REFERENCE

.NET client libraries

Integrate with Slack or Microsoft Teams

HOW-TO GUIDE

Azure Boards with Slack

Azure Pipelines with Slack

Azure Pipelines with Microsoft Teams

Integrate with Service hooks

OVERVIEW

Integrate with service hooks

TUTORIAL

Create a service hook with Microsoft Teams

Create a service hook with WebHooks

Extensions overview

Article • 11/28/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Extensions are add-ons, which you can use to customize and extend your experience with Azure DevOps. They're written with standard technologies - HTML, JavaScript, CSS - and developed using your preferred development tools.

Extensions get published at the Visual Studio Marketplace, where they can be kept private for you and your team or shared publicly with the millions of developers currently using Azure DevOps.

Extensions use our RESTful API Library to easily interact with Azure DevOps and applications/services.

Understand parts of an extension

The following items make up an extension:

- JSON manifest file: Contains basic info about the extension.
- Discovery assets: Markdown and images that make up the extension's overview and aesthetics in the Marketplace.
- Static files: Contain the logic of the extension, including HTML, JS, and CSS files. Static files are only applicable to contribution-based extensions.

These files and assets get bundled up to make a VSIX file that gets published to the Marketplace.

From the Marketplace, users can install extensions directly into their organization. If you don't have permissions to install an extension, but you're a project member, you can

request an extension instead.

Use an extension

There are dozens of ways you can use an extension and places where you can add to the user interface, and we're adding more every sprint. Learn about all of the places where you can add a hub in the Extensibility points.

- Provide new Azure Pipelines tasks that teams can use in their builds.
- Use dashboard widgets to get custom views within Azure DevOps.
- Extend the work item form with new tabs, sections, and actions.
- Create your own hub to embed new capabilities within our Agile, code, build, and test experiences.
- Develop actions that can be run on hubs, whether they're ours or ones you created.

Evaluate a Marketplace extension

To evaluate a Marketplace extension, review the information and resources described in the following table. You can find this information in the extension information

Information

Usage

Top Publisher badge

The publisher demonstrates commitment to its customers and the Marketplace through excellent policies, quality, reliability, and support. For more information, see Top Publisher.

Q & A

The Q & A section of published extensions might answer questions you have. Also, they're a good mechanism to engage with the extension's publisher(s) to have a meaningful dialogue to make yourself comfortable. Use the Q & A information to

understand the development, testing, and security practices the publisher follows. It also gives you a sense of the publisher's responsiveness.

Ratings & reviews

Ratings and reviews indicate how others perceive the offering. For more information, see Respond to customer feedback.

Privacy, license, and support policies

See if the publisher provided them and if they meet your needs or concerns. For more information, go to Safety information.

Safety information

- **Malware scan**: The Marketplace runs a virus scan on each new and updated extension package that publishes to ensure its safety. Until the scan is all clear, we don't publish the extension for public usage. If a concern surfaces, the Marketplace team has the means to disable an extension immediately and notify its existing customers.
- **Content scan**: The Marketplace scans content for every new and updated extension to avoid surfacing inappropriate or offensive content on the Marketplace pages.
- **Access to approved scopes only**: An extension can only operate within the granted scopes. For example, an extension that has only read permissions on work items can't modify your features and bugs. Azure DevOps web extensions run in a sandboxed browser IFrame. They're only able to access Azure DevOps data and APIs approved for the extension. During installation, admins are prompted to approve permissions and scopes. One way to protect yourself is to carefully review the scopes the extension requests.

  > **Note**
  >
  > If the scopes change for an extension, the you must approve the update before it can become applicable on your organization or collection.

- **Third-party build and release tasks**: Tasks are implemented as code that executes on an agent machine. Tasks are only able to access secrets explicitly provided to them (see variable secrets), but tasks generally have full access to the agent

machine itself. To reduce risk, run builds on Microsoft-hosted agents, which are VMs isolated from other jobs and recycled after each job. Or, limit file and network access on private hosted agent machines. Learn more about build and release agents.

- **Third-party code execution on the server**: There's no way for an extension to install or execute any code on Azure DevOps Server.

Build an extension

Before you build an extension, familiarize yourself with the extension types already available within the Marketplace, Extensions for Azure DevOps. Learn how to build your first extension and check out our full set samples.

- Develop your first extension
- Samples

For more information about building extensions, see the following articles:

- REST APIs
- Service Hooks
- Package, publish, and install your extension
- Package and publish your integration with an external app or service
- Share your work publicly with the entire community

Next steps

Develop an extension

Related articles

- Visual Studio Marketplace
- Extension Publisher Page
- Visual Studio Partner Program
- Extension manifest reference

Install extensions

Article • 10/09/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Install, assign, disable, and uninstall extensions, which add new features and capabilities for Azure DevOps.

For more information about extensions, see the developing and publishing overviews.

Prerequisites

- **Permissions:** Be a member of the Project Collection Administrators group to install extensions. Organization owners are automatically members of this group. If you don't have permissions, you can request extensions instead or look up a project collection administrator.
- **Extension sharing:** Private extensions must be shared with your organization to be installed. For more information about sharing private extensions, see the publishing documentation.

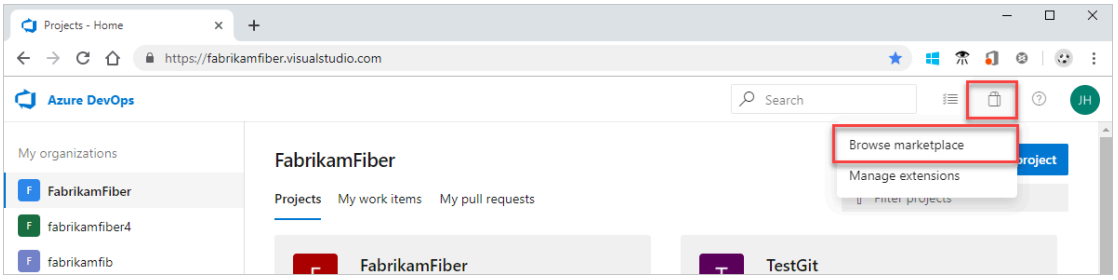
Install an extension

Install an extension to your organization by doing the following steps.

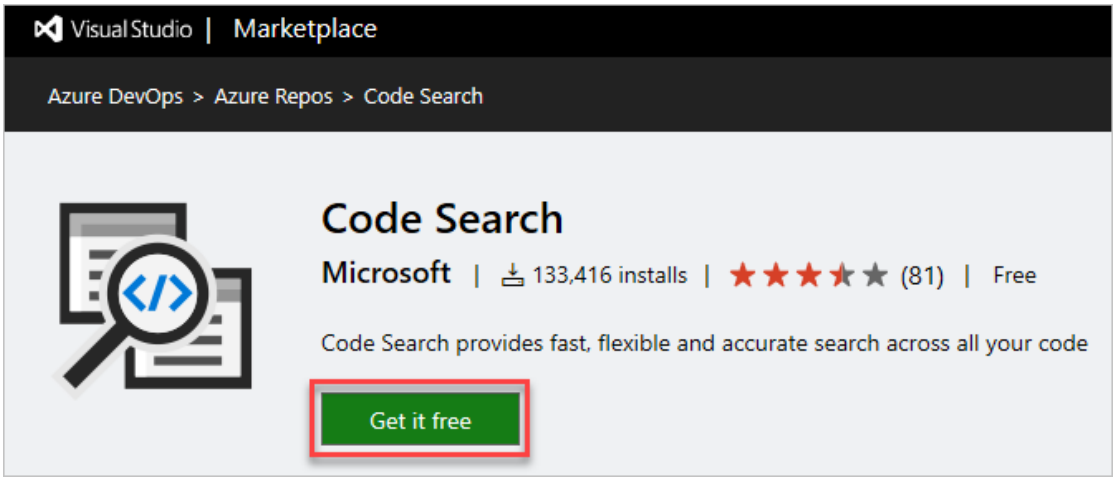
Browser

1. Sign in to your organization (`https://dev.azure.com/{yourorganization}`).

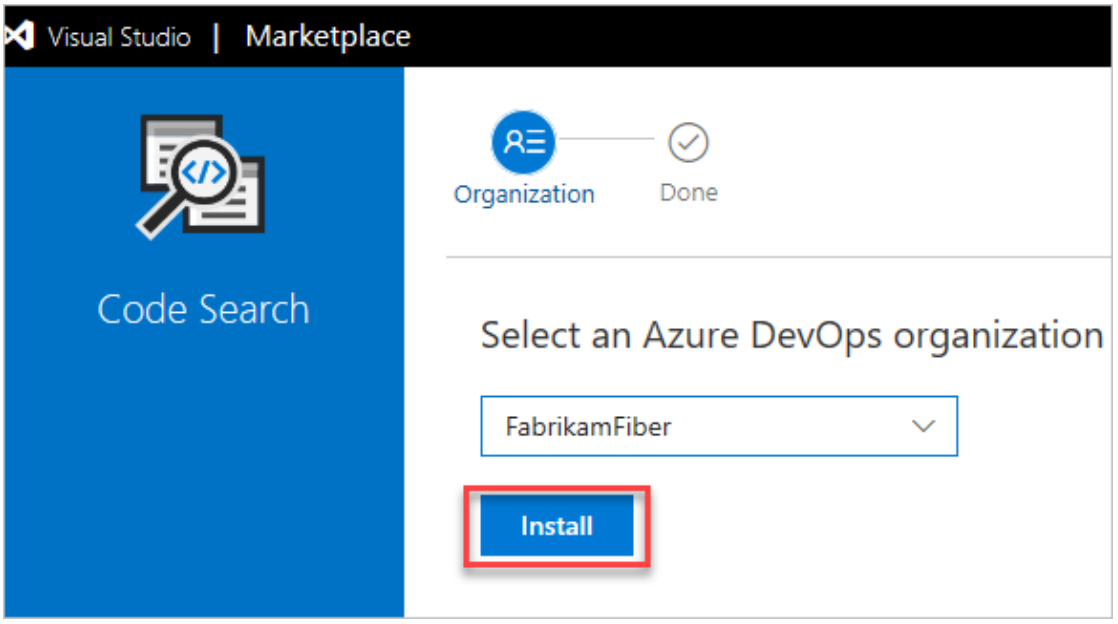
2. Select the shopping bag icon, and then select **Browse Marketplace**.



3. Find the extension that you want to install and select **Get it free**.

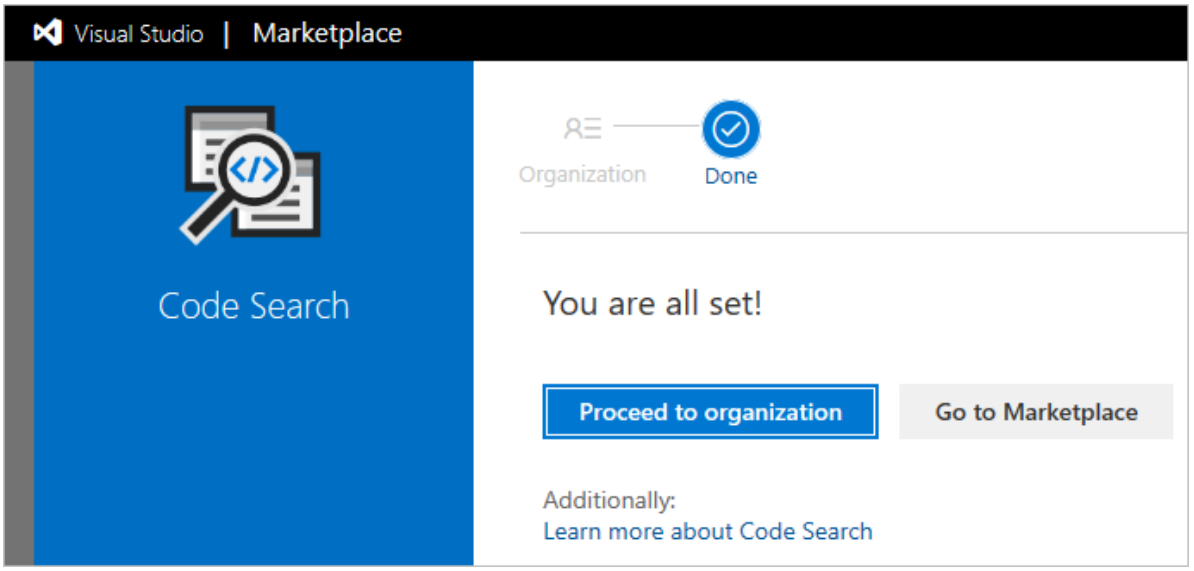


4. Select your organization from the dropdown menu, and then select **Install** to install the extension.



- Why don't I see any organizations?
- Why can't I install this extension?

Your extension is now installed! You can now go to your organization to use your extension. Also, tell your team about this extension, so they can start using its capabilities.

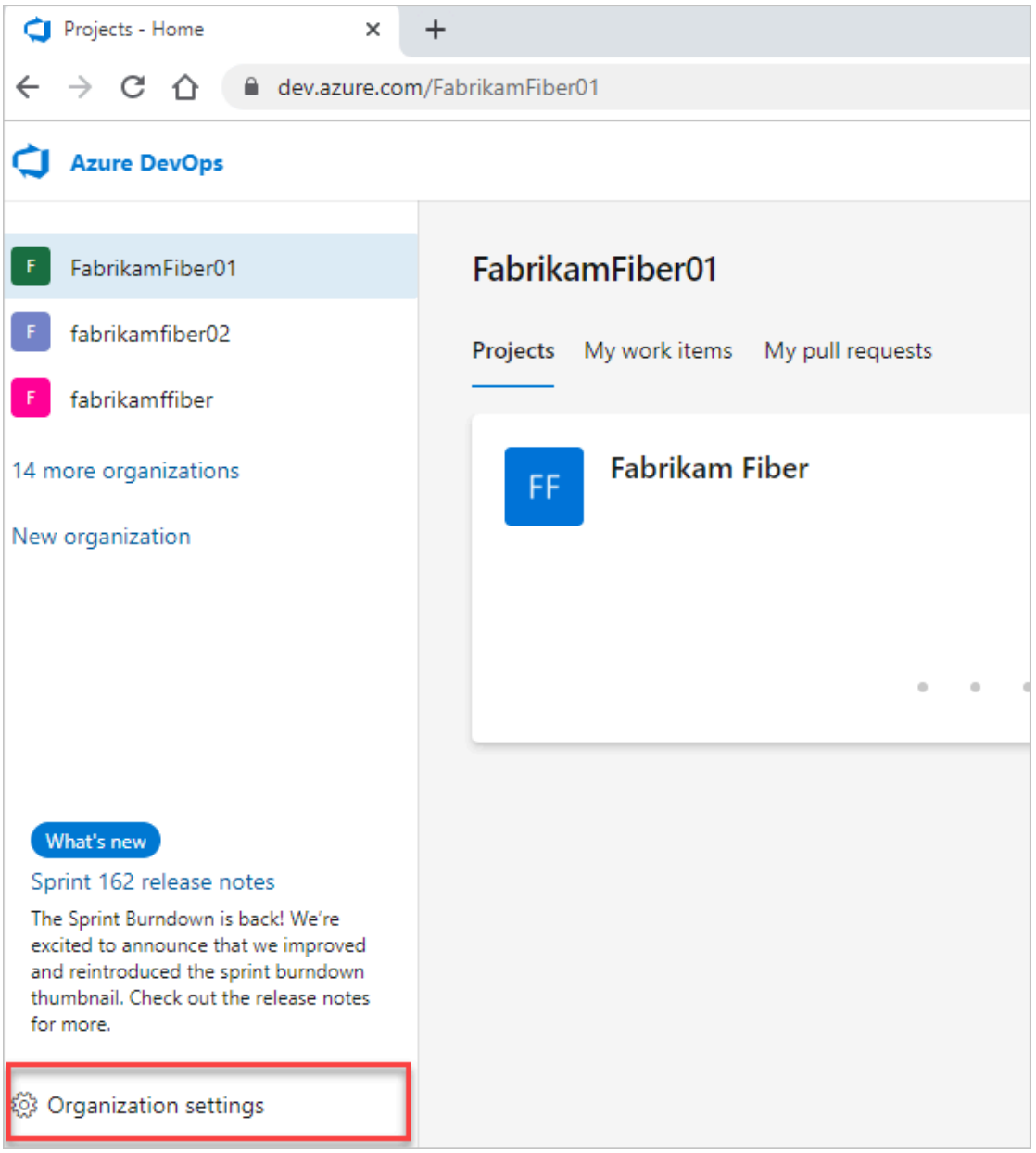


Uninstall or disable an extension

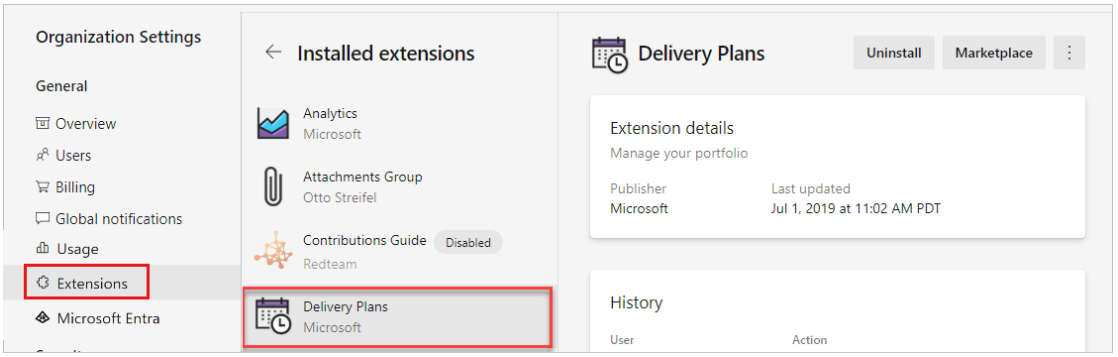
Browser

1. Sign in to your organization (`https://dev.azure.com/{yourorganization}`).

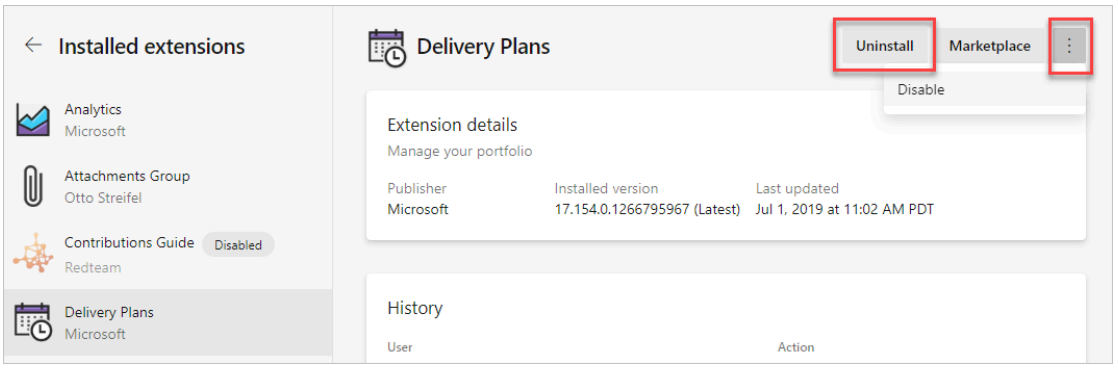
2. Select **Organization settings**.



3. Select **Extensions**, and then select the extension that you want to uninstall or disable.



4. Select **Uninstall** or select the ellipses (...), and then select **Disable**.



Enable or list extensions through the command line

Enable an extension with the az devops extension enable command. To get started, see Get started with Azure DevOps CLI.

Azure CLI

```
az devops extension enable --extension-name
                           --publisher-name
                           [--org]
```

Parameters - enable extension

- **extension-name**: The name of the extension to enable.
- **publisher-name**: The name of the extension publisher.
- **org**: Azure DevOps organization URL. Configure the default organization with `az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using `git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.

Example - enable extension

The following command enables the **Timetracker** extension and shows the result in table format.

Azure CLI

```
az devops extension enable --extension-name Timetracker --publisher-name
7pace --output table
```

```
Publisher Id    Extension Id    Name         Version      Last Updated
States
--------------  --------------  -----------  -----------  ---------------  -
-------
7pace           Timetracker     Timetracker  5.0.1.34507  2019-11-13
none
```

List extensions

You can list the extensions that are installed in your organization with the az devops extension list command. To get started, see Get started with Azure DevOps CLI.

Azure CLI

```
az devops extension list [--include-built-in {false, true}]
                         [--include-disabled {false, true}]
                         [--org]
```

Optional parameters - list extensions

- **include-built-in**: Include the built-in extensions. Accepted values are *true* (default) and *false*.
- **include-disabled**: Include the disabled extensions. Accepted values are *true* (default) and *false*.
- **org**: Azure DevOps organization URL. You can configure the default organization using `az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using `git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.

Example - list extensions

The following command lists extensions in your organization. It excludes the disabled and built-in extensions, and shows the results in table format.

Azure CLI

```
az devops extension list --include-built-in false --include-disabled false -
output table

Publisher Id    Extension Id             Name                      Version
Last Updated     States                  Flags
--------------  ----------------------  ----------------------  ----------
---------  ---------------  ----------------------  -------
ms              vss-analytics            Analytics
```

```
18.160.0.2130149925  2019-11-22       multiVersion, truste...  trusted
ms                   vss-code-search     Code Search
18.160.0.1640944814  2019-11-22       multiVersion, truste...  trusted
ms                   vss-plans           Delivery Plans
18.160.0.1266795967  2019-11-25       multiVersion, truste...  trusted
ms-eswm              dependencytracker   Dependency Tracker
2.1910.12801         2019-10-28       none
ms-devlabs           workitem-feature-tim...  Feature timeline and...  0.0.357
2019-10-14           none
AgileParts           gantt               GANTT chart              1.0.79
2019-10-25           none
gordon-bee...        github              GitHub Widget            0.10.0
2016-03-16           none
ms-devlabs           vsts-extensions-mult...  Multivalue control       2.2.26
2019-11-15           none
agile-exte...        product-vision      Product Vision           2.0.6
2019-06-04           none
mohitbagra           related-workitems   Related Work items       2.0.4
2017-11-12           none
YodLabs              TagsManager2        Tags Manager             0.9.31
2019-02-04           none
ms-devlabs           team-calendar       Team Calendar            2.0.15
2019-11-01           none
ms                   vss-testmanager-web     Test Manager for TFS...
18.160.0.2130893445  2019-11-25       multiVersion, truste...  trusted
mmanela              vsts-workitem-recent...  Who recently viewed ...  1.0.4
2019-03-22           none
ottostreif...        wiql-editor         Wiql Editor              2.0.90
2019-06-21           none
mohitbagra           workitem-checklist  Work item checklist      3.2.4
2019-06-24           none
mohitbagra           witoneclickactions  Work item form one c...  2.3.2
2018-04-03           none
ms-devlabs           WorkItemVisualizatio...  Work Item Visualizat...  1.4.64
2018-04-03           none
```

List extension information

You can list the details about an extension with the az devops extension show command. To get started, see Get started with Azure DevOps CLI.

Azure CLI

```
az devops extension show --extension-name
                         --publisher-name
                         [--org]
```

Parameters - list extension information

- **extension-name**: The name of the extension.
- **publisher-name**: The name of the extension publisher.
- **org**: Azure DevOps organization URL. You can configure the default organization using `az devops configure -d organization=ORG_URL`. Required if not configured as default or picked up using `git config`. Example: `--org https://dev.azure.com/MyOrganizationName/`.

Example - list extension information

The following command shows information about the **Timetracker** extension in table format.

Azure CLI

```
az devops extension show --extension-name Timetracker --publisher-name 7pace
--output table

Publisher Id    Extension Id    Name         Version      Last Updated
States
--------------  --------------  -----------  -----------  ---------------  -
-------
7pace           Timetracker     Timetracker  5.0.1.34507  2019-11-13
disabled
```

Troubleshoot extension installation

Follow these troubleshooting steps to resolve common issues:

- **Extension fails to install:**
  - **Check permissions:** Ensure you have the necessary permissions to install extensions. You need to be a Project Collection Administrator or have the necessary permissions granted by an administrator.
  - **Verify extension compatibility:** Ensure the extension is compatible with your version of Azure DevOps. Check the extension's details page for compatibility information.
  - **Network issues:** Verify that your network connection is stable and that there are no firewall or proxy settings blocking the installation process.
- **Extension crashes or causes errors:**
  - **Collect diagnostic information:** If an extension crashes or causes errors, collect diagnostic information to help identify the issue. This information includes error messages, logs, and screenshots of the problem.

    - **Check extension logs:** Some extensions provide logs that can help diagnose issues. Check the extension's documentation for information on how to access and interpret these logs.
    - **Contact support:** If you can't resolve the issue, contact the extension's support team with the collected diagnostic information. Provide as much detail as possible to help them troubleshoot the problem.

- **Extension not visible after installation:**
    - **Refresh the page:** Sometimes, the extension might not appear immediately after installation.
    - **Check permissions:** Ensure you have the necessary permissions to view and use the extension. Some extensions might require specific permissions to be visible.
    - **Reinstall the extension:** If the extension still isn't visible, try uninstalling and reinstalling it.

Next steps

Manage extension permissions

Related articles

- Request extensions and approve extension requests
- Develop a web extension
- Azure Billing and Subscription FAQs
- Azure billing support
- Visual Studio subscriptions support

ⓘ **Note:** The author created this article with assistance from AI. Learn more

Feedback

Was this page helpful? Yes No

Provide product feedback

Request and approve extensions

Article • 01/28/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

If you don't have permissions to install extensions, you can request extensions instead. Project Collection Administrators receive an email when another project member requests an extension. Azure DevOps automatically installs the extension once it's approved.

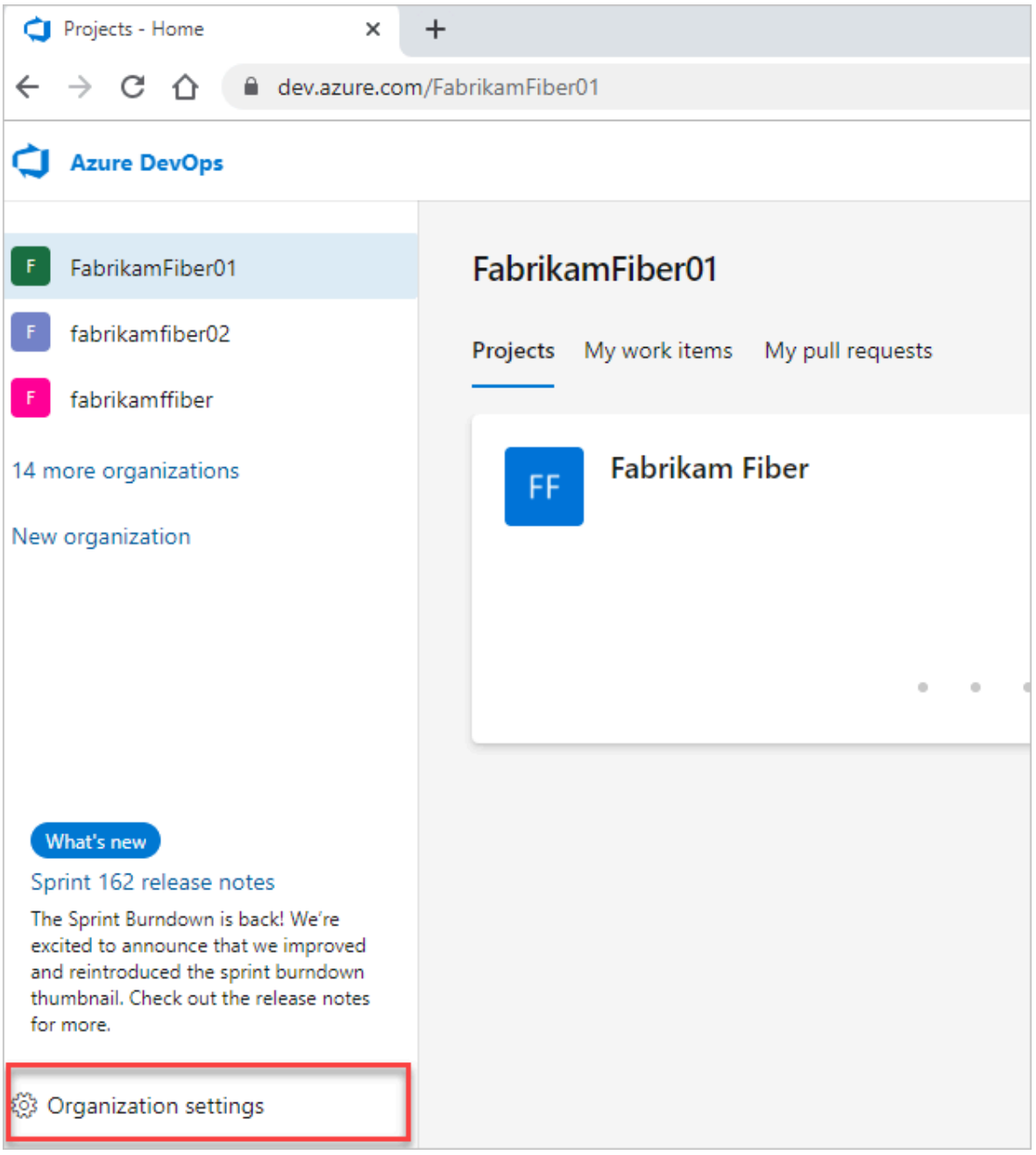
Prerequisites

Permissions:

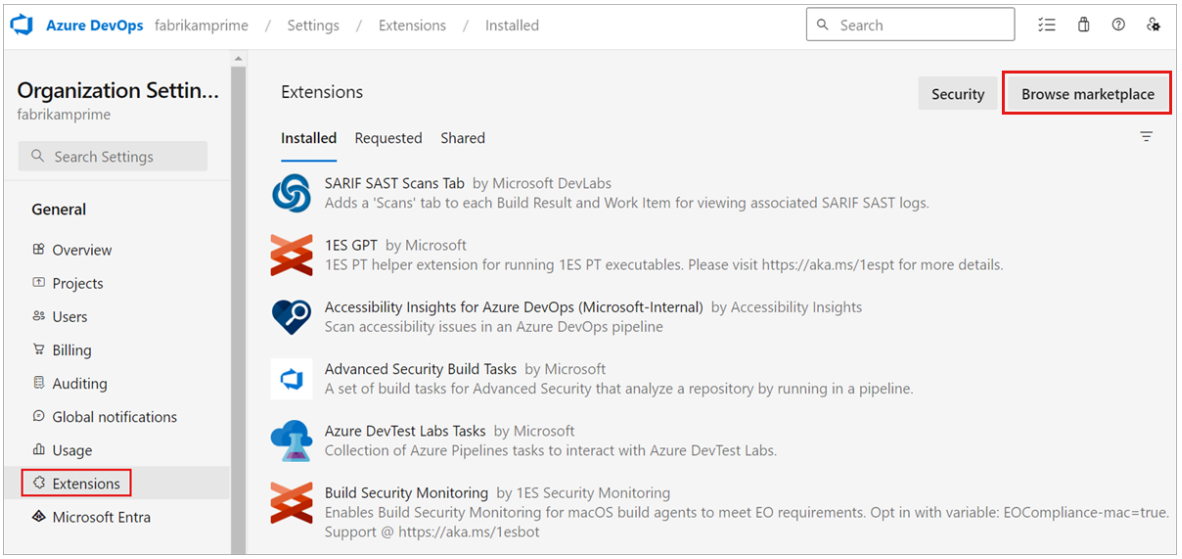
- To request extensions: Contributor for your organization.
- To approve extensions: Member of the Project Collection Administrators group and **Edit collection-level information** permissions.

Request an extension

1. Sign in to your organization (`https://dev.azure.com/{yourorganization}`).
2. Select **Organization settings**.



3. Select **Extensions**, and then **Browse marketplace**.

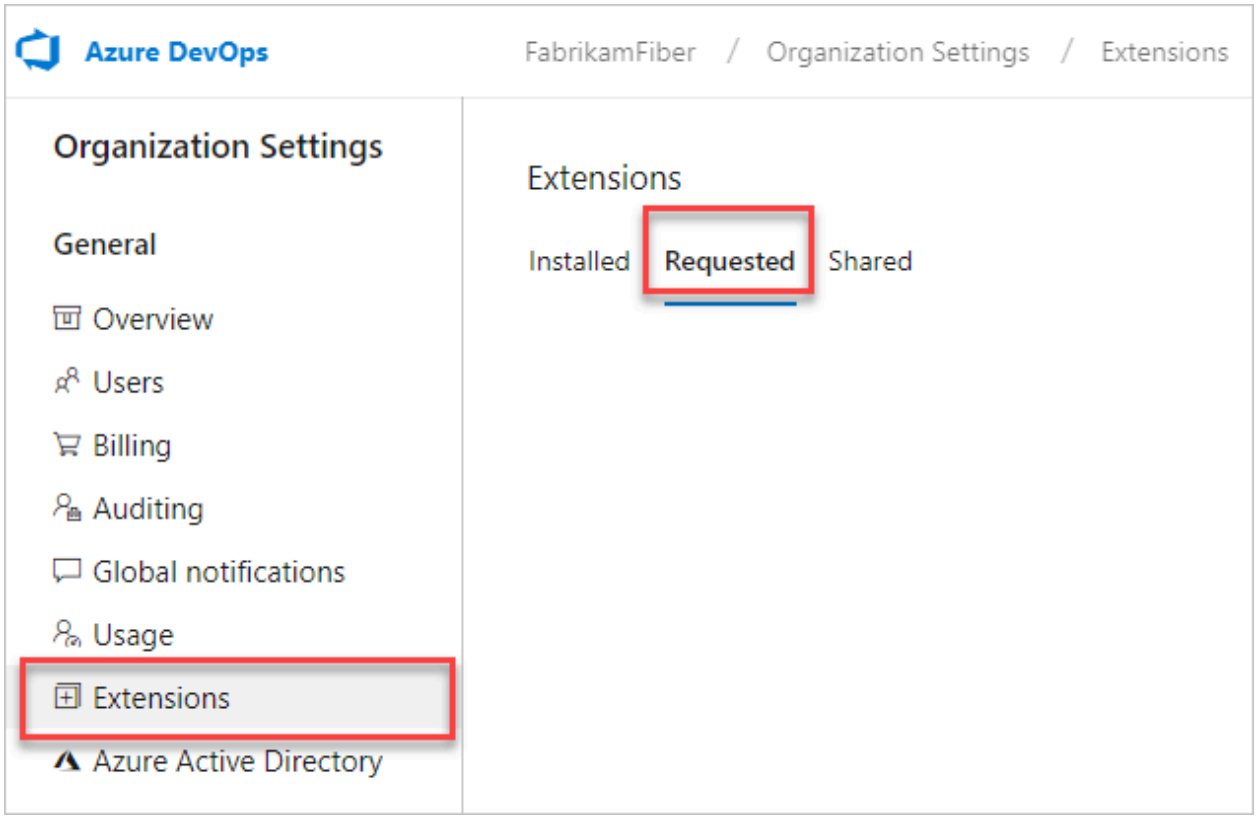


4. Select an extension to install.

5. If you don't have permission to install the extension, you can request it now.

Review your requests after the Marketplace sends the request to your Project Collection Administrator.

Your requests appear on the **Extensions** page, **Requested** tab.

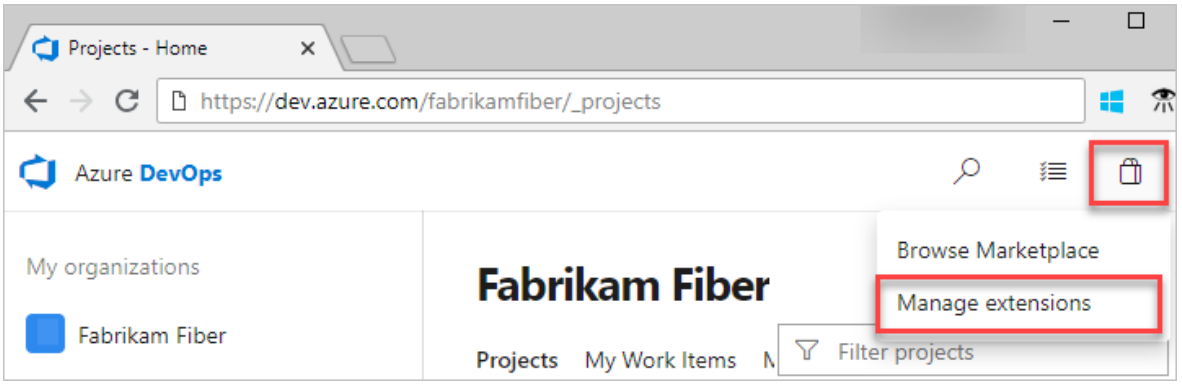


To approve extensions, have **edit collection-level information** permissions.

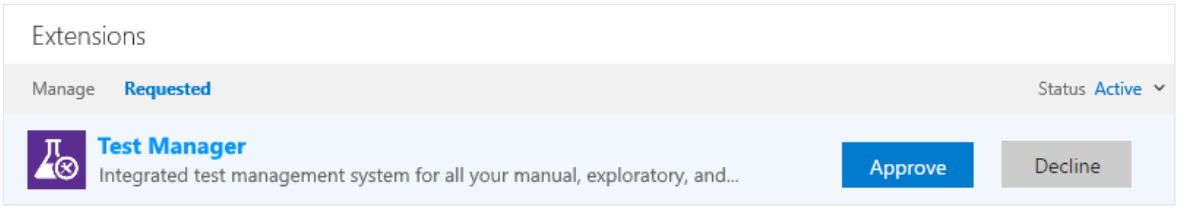
Approve an extension request

1. Go to your Azure DevOps project, `https://dev.azure.com/{organization}/{project}`.

2. Select the shopping bag icon, and then **Manage extensions**.



3. Review and approve your requested extensions.



After you approve extension requests, the extensions are automatically installed.

4. Assign those extensions to users who need access.

Tell your team about installed extensions, so they can start using their capabilities.

Related articles

- FAQs
- Set up billing
- Azure DevOps pricing
- Assign access levels by group membership

Feedback

Was this page helpful? Yes No

Provide product feedback

Manage extension permissions

Article • 01/28/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

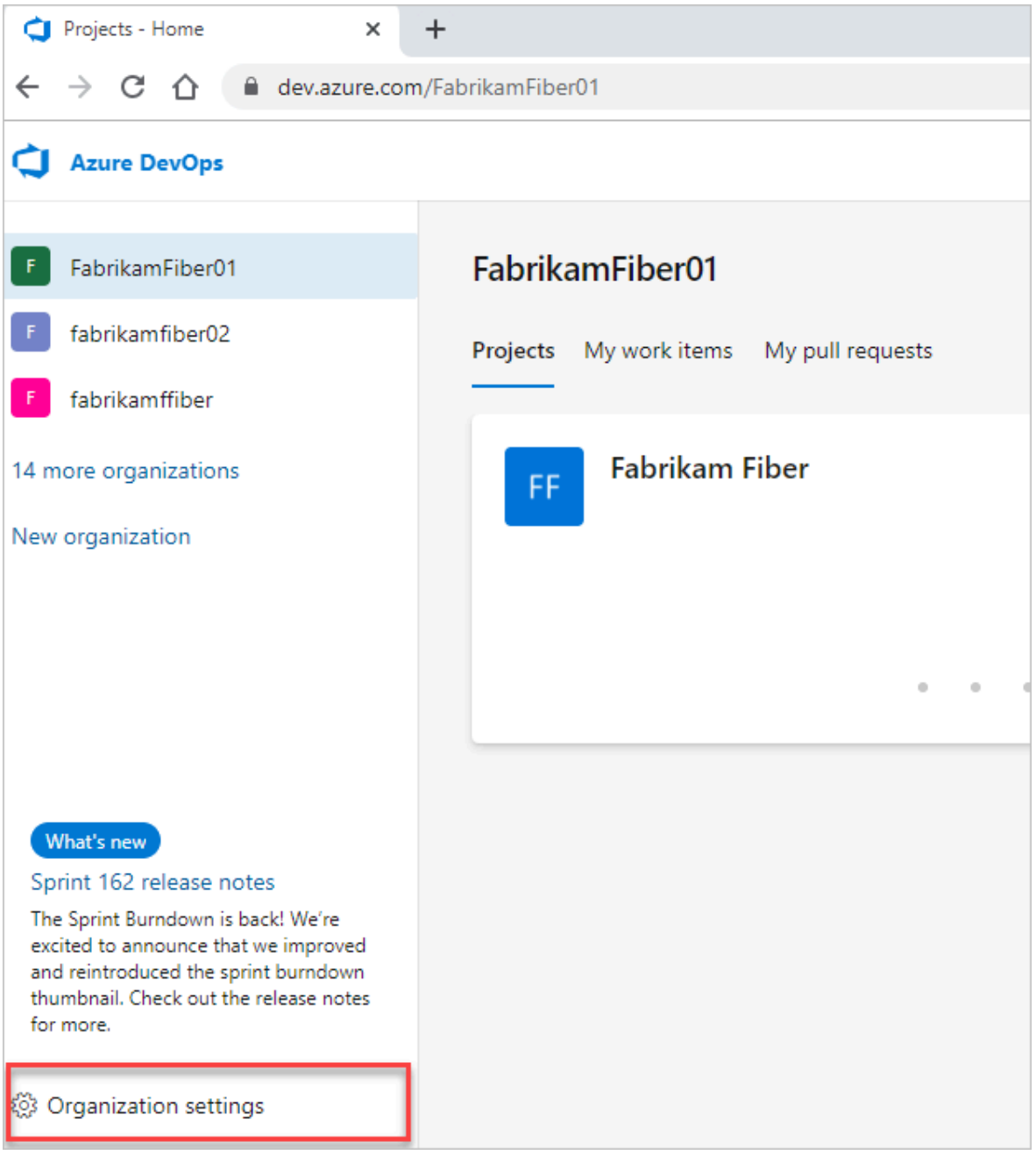
Learn how to manage permissions for users or groups, so they can manage extensions.

Prerequisites

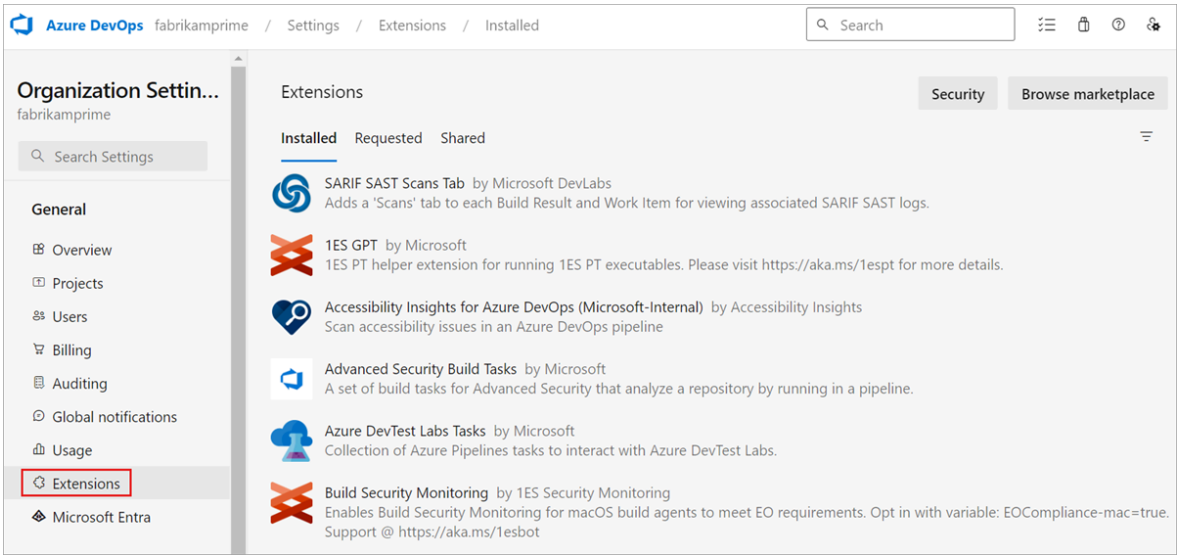
- **Permissions:** Member of the Project Collection Administrators group. Organization owners are automatically members of this group.
- **Extension sharing:** Private extensions shared with your organization.

Manage permissions

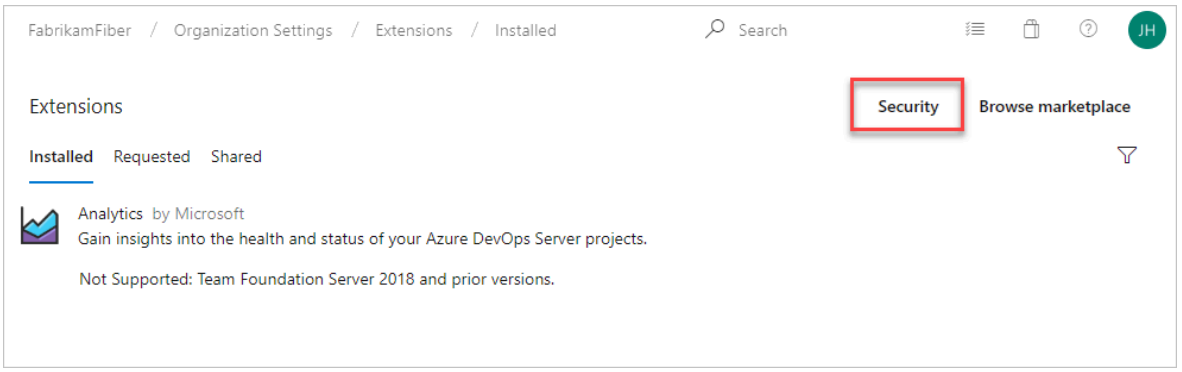
1. Sign in to your organization (`https://dev.azure.com/{yourorganization}`).
2. Select **Organization settings**.



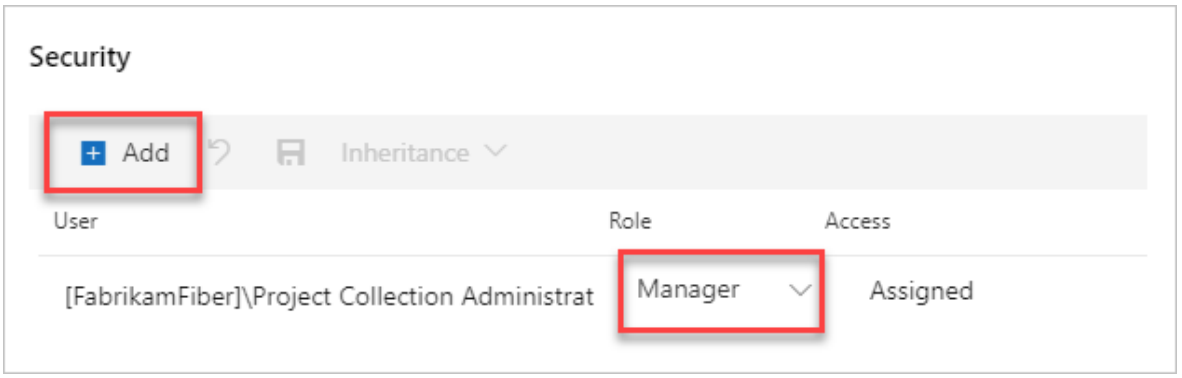
3. Select **Extensions**.



4. Select **Security**.



5. Add users or update permission settings.



Related articles

- Install extensions
- Request extensions
- Uninstall or disable extensions
- About permissions

Feedback

Was this page helpful? Yes No

Provide product feedback

Choose the right authentication mechanism

Article • 01/07/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

For applications that interface with Azure DevOps Services, you must authenticate to gain access to resources like REST APIs. This article provides guidance to help you choose the right authentication mechanism for your application.

The following table outlines suggested authentication concepts to consider for different application scenarios. Refer to the accompanying descriptions, examples, and code samples to help get you started.

Expand table

Type of application	Description	Example	Authentication mechanism	Code samples
Interactive client-side app (REST)	Client application that allows user interaction calling Azure DevOps Services REST APIs	Console application enumerating projects in an organization	OAuth with Microsoft Authentication Library (MSAL)	sample
Interactive client-side app (client libraries)	Client application that allows user interaction calling Azure DevOps Services client libraries	Console application enumerating bugs assigned to the current user	OAuth with client libraries	sample
Non-interactive client-side app	Headless text only client-side application	Console app displaying all bugs assigned to a user	OAuth with Device Profile flow	sample
Personal access token (PAT)	Bearer token to access your own resources	Use your PAT in place of your password for ad-hoc REST calls. Not ideal for applications.	PATs	examples
Server app	Azure DevOps Server app using the Client OM library	Azure DevOps Server extension displaying team bug dashboards	Client Libraries	sample

Type of application	Description	Example	Authentication mechanism	Code samples
Service principal or Managed identity	Application with its own identity	Azure function to create work items	Service principals and managed identities	sample
Web extension	Azure DevOps Services extension	Agile Cards extension	VSS Web Extension SDK	sample

> **Tip**
>
> Entra-based authentication is our recommendation for developers looking to integrate with Azure DevOps Services, if you are interacting with Microsoft Entra accounts. The OAuth sample apps in this table are using Microsoft Entra's identity platform for app development.
> For authentication with Microsoft accounts (MSA) or Azure DevOps Server users, look into our client libraries or PATs.
> Read up more in our blog on how we're reducing PAT usage across our platform.

Frequently asked questions (FAQs)

Q: Why can't my service account access the Azure DevOps REST API?

A: Your service account might not have "materialized." Service accounts without interactive sign-in permissions can't sign in. For more information, see this work-around for a solution.

Q: Should I use Azure DevOps Services Client Libraries or Azure DevOps Services REST APIs for my interactive client-side application?

A: We recommend using Azure DevOps Services Client Libraries over REST APIs for accessing Azure DevOps Services resources. They're simpler and easier to maintain when REST endpoint versions change. If the client libraries lack certain functionality, use MSAL for authentication with our REST APIs.

Q: Is this guidance only for Azure DevOps Services or is it also relevant for on-premises Azure DevOps Server users?

A: This guidance is primarily for Azure DevOps Services users. For Azure Devops Server users, we recommend using the Client Libraries, Windows Authentication, or Personal Access Tokens (PATs) for authentication.

Q: What if I want my application to authenticate with both Azure DevOps Server and Azure DevOps Services?

A: The best practice is to have separate authentication paths for Azure DevOps Server and Azure DevOps Services. You can use the `requestContext` to determine which service you're accessing and then apply the appropriate authentication mechanism. If you prefer a unified solution, PATs work for both.

Related resources

- About security and identity.
- Azure DevOps data protection overview
- Sample applications

Feedback

Was this page helpful? Yes No

Provide product feedback

Authenticate to Azure DevOps with Microsoft Entra

Article • 01/31/2025

Microsoft Entra ID is a separate Microsoft product with its own platform. As a leading identity and access management (IAM) provider, Microsoft Entra ID focuses on managing team members and safeguarding company resources. You can connect your Azure DevOps organization to a Microsoft Entra ID tenant, which offers many benefits to your company.

Once connected, the Microsoft Identity application platform on top of Microsoft Entra ID provides several advantages for app developers and org admins. You can register an application to access Azure tenants and define permissions needed from Azure resources, including Azure DevOps, which exists outside of the Azure tenant construct.

Microsoft Entra apps and Azure DevOps apps are separate entities with no knowledge of each other. The authentication methods differ: Microsoft Entra uses OAuth, while Azure DevOps uses its own OAuth. Microsoft Entra ID OAuth apps issue Microsoft Entra tokens, not Azure DevOps access tokens. These tokens have a standard one-hour duration before expiration.

Develop Azure DevOps apps on Microsoft Entra

Read the Microsoft Entra documentation thoroughly to understand the new functionality and different expectations during setup.

We support your app development with guidance for:

- Microsoft Entra OAuth apps (on-behalf-of user apps) for apps that perform actions on behalf of consenting users.
- Microsoft Entra service principals and managed identities (on-behalf-of itself apps) for apps that perform automated tooling within a team.

Replace PATs with Microsoft Entra tokens

Personal access tokens (PATs) are popular for Azure DevOps authentication due to their ease of creation and use. However, poor PAT management and storage can lead to unauthorized access to your Azure DevOps organizations. Long-lived or over-scoped PATs increase the risk of damage from a leaked PAT.

Microsoft Entra tokens offer a secure alternative, lasting only one hour before requiring a refresh. The authentication protocols for generating Entra tokens are more robust and secure. Security measures like conditional access policies protect against token theft and replay attacks. We encourage users to explore using Microsoft Entra tokens instead of PATs. We share popular PAT use cases and ways to replace PATs with Entra tokens in this workflow.

Ad-hoc requests to Azure DevOps REST APIs

You can also use the **Azure CLI** to get Microsoft Entra ID access tokens for users to call Azure DevOps REST APIs. Since Entra access tokens only last for one hour, they're ideal for quick one-off operations, like API calls that don't need a persistent token.

Acquire user tokens

Azure CLI

1. Sign in to the Azure CLI using the `az login` command and follow the on-screen instructions.
2. Set the correct subscription for the signed-in user with these bash commands. Make sure the Azure subscription ID is associated with the tenant connected to the Azure DevOps organization you're trying to access. If you don't know your subscription ID, you can find it in the Azure portal.

   Bash

   ```
   az account set -s <subscription-id>
   ```

3. Generate a Microsoft Entra ID access token with the `az account get-access-token` command using the Azure DevOps resource ID: `499b84ac-1321-427f-aa17-267ca6975798`.

   Bash

   ```
   az account get-access-token \
   --resource 499b84ac-1321-427f-aa17-267ca6975798 \
   --query "accessToken" \
   -o tsv
   ```

For more information, see the Databricks docs.

Acquire service principal tokens in Azure CLI

Service principals can also use ad-hoc Microsoft Entra ID access tokens for ad-hoc operations. For more information, see Service principals and managed identities/Get a Microsoft Entra ID token with the Azure CLI.

Git operations with Git Credential Manager

You can also use Microsoft Entra tokens to perform Git operations. If you regularly push to git repositories, use the Git Credential Manager to easily request and manage your Microsoft Entra OAuth token credentials, as long as `oauth` is set as the default `credential.azReposCredentialType`.

Related articles

- Build Azure DevOps integration with Microsoft Entra OAuth apps
- Use service principals & managed identities in Azure DevOps

---

Feedback

Was this page helpful? Yes No

Provide product feedback

Use service principals & managed identities in Azure DevOps

Article • 01/08/2025

Azure DevOps Services

> **Note**
>
> Azure Active Directory (Azure AD) is now Microsoft Entra ID. For more information, see New name for Azure AD.

Add Microsoft Entra service principals and managed identities as application identities into your Azure DevOps Services organizations to grant them access to your organization resources. For many teams, this feature can be a viable and preferred alternative to personal access tokens (PATs) when you authenticate applications that power automation workflows in your company.

About service principals and managed identities

Service principals are security objects within a Microsoft Entra application that define what an application can do in a given tenant. They're set up in the Azure portal during the application registration process and configured to access Azure resources, like Azure DevOps. By adding service principals into your organization and setting up permissions on top of them, we can determine whether a service principal is authorized to access your organizational resources and which ones.

Managed identities is another Microsoft Entra feature that acts similarly to an application's service principals. These objects provide identities for Azure resources and allow an easy way for services that support Microsoft Entra authentication to share credentials. They're an appealing option because Microsoft Entra ID takes care of credential management and rotation. While setup for a managed identity might look different on the Azure portal, Azure DevOps treats both security objects the same as a new application identity in an organization with defined permissions. Throughout the rest of this article, we refer to managed identities and service principals interchangeably as service principal, unless specified.

Use the following steps to authenticate these identities to Azure DevOps to allow them to perform actions on behalf of themselves.

Configure managed identities and service principals

Your implementation might vary, but at a high-level, the following steps help you start using service principals in your workflow. To follow along, consider looking at one of our sample apps.

1. Create a new managed identity or application service principal

Create an application service principal or a managed identity in the Azure portal.

Option 1: Create an application service principal

When you create a new application registration, an application object is created in Microsoft Entra ID. The **application service principal** is a representation of this application object for a given tenant. When you register an application as a multitenant application, there's a unique service principal object that represents the application object for every tenant the application is added to.

Further information:

- Application and service principal objects in Microsoft Entra ID
- Securing service principals
- Use the portal to create a Microsoft Entra application and service principal that can access resources

Option 2: Create a managed identity

Creating managed identities in the Azure portal differs significantly from setting up applications with service principals. Before you begin the creation process, you must first consider which type of managed identity you want to create:

- **System-assigned managed identity:** Some Azure services allow you to enable a managed identity directly on a service instance. When you enable a system-assigned managed identity, an identity is created in Microsoft Entra ID. The identity is tied to the lifecycle of that service instance. When the resource is deleted, Azure automatically deletes the identity for you. By design, only that Azure resource can use this identity to request tokens from Microsoft Entra ID.
- **User-assigned managed identity** You might also create a managed identity as a standalone Azure resource by creating a user-assigned managed identity and

assign it to one or more instances of an Azure service. For user-assigned managed identities, the identity is managed separately from the resources that use it.

For more information, see the following articles and video:

- What are managed identities for Azure resources?
- Manage user-assigned managed identities
- Configure managed identities for Azure resources on a VM using the Azure portal

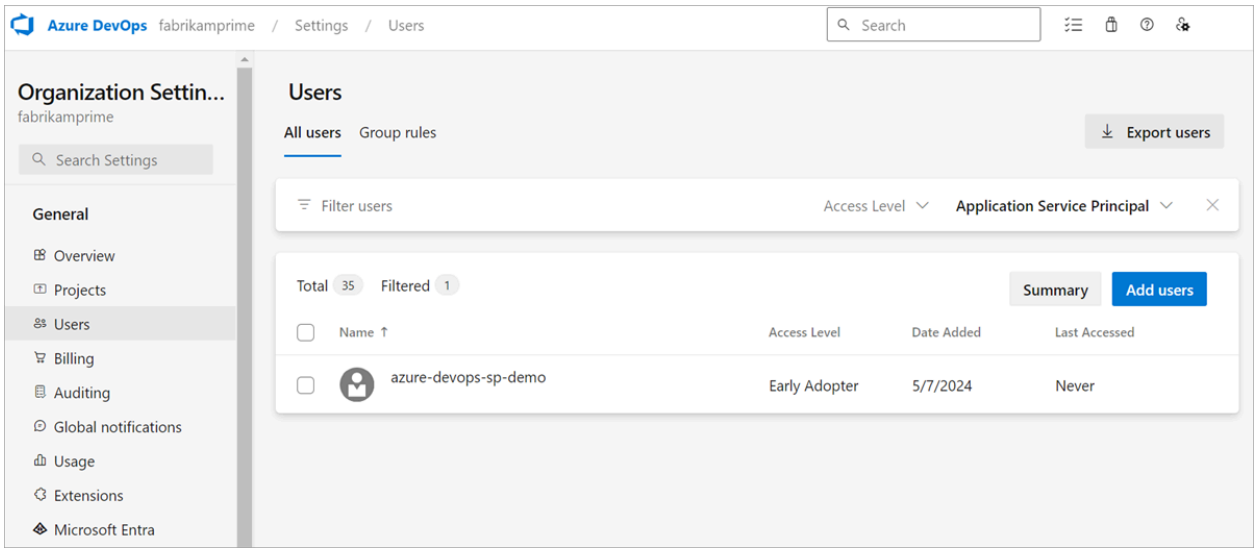
https://learn-video.azurefd.net/vod/player?id=b9cf4e49-db98-4247-8770-b8707b043f17&locale=en-us&embedUrl=%2Fazure%2Fdevops%2Fintegrate%2Fget-started%2Fauthentication%2Fservice-principal-managed-identity

2. Add a service principal to an Azure DevOps organization

Once you configure the service principals in the Microsoft Entra admin center, you must do the same in Azure DevOps by adding the service principals to your organization. You can add them through the Users page or with the ServicePrincipalEntitlements APIs. Since they can't sign in interactively, a user account that can add users to an organization, project, or team must add them. Such users include **Project Collection Administrators** (PCA) or **Project Administrators and Team Administrators** when the "Allow team and project administrators to invite new users" policy is enabled.

> **Tip**
>
> To add a service principal to the organization, enter the application or managed identity's display name. If you choose to add a service principal programmatically through the **ServicePrincipalEntitlements API**, make sure to pass in the **service principal's object id** and not the application's object ID.

If you're a PCA, you can also grant a service principal access to specific projects and assign a license. If you're not a PCA, you must reach out to the PCA to update any project memberships or license access levels.



> **Note**
>
> You can only add a managed identity or service principal for the tenant your organization is connected to. Service principals can be made multitenant to access multiple tenants at once. Managed identities can only belong to a single tenant. To access a managed identity in a different tenant, see the workaround in the FAQ.

3. Set permissions on a service principal

After your service principals are added to the organization, you can treat them similarly to standard user accounts. You can assign permissions directly on a service principal, add it to security groups and teams, assign it to any access level, and remove it from the organization. You can also use the Service Principal Graph APIs to perform CRUD operations on service principals.
https://learn-video.azurefd.net/vod/player?id=e8ee6f58-3939-4170-b112-0ca1b39b0298&locale=en-us&embedUrl=%2Fazure%2Fdevops%2Fintegrate%2Fget-started%2Fauthentication%2Fservice-principal-managed-identity

Setting these permissions might differ from how you're used to setting up application permissions in a Microsoft Entra application for other Azure resources. Azure DevOps doesn't rely on the "Application permissions" setup available to application registrations through the Azure portal. These application permissions apply permissions to a service principal across all organizations tied to a tenant and have no knowledge of the organization, project, or object permissions available in Azure DevOps. To offer service principals more granular permissions, we rely on our own permissions model instead of Microsoft Entra IDs.

4. Manage a service principal

Management of service principals differs from user accounts in the following key ways:

- Service principals don't have emails and as such, they can't be invited to an organization via email.
- Group rules for licensing currently don't apply to service principals. If you want to assign an access level to a service principal, it's best to do so directly.
- Service principals can be added to Microsoft Entra groups (in the Azure portal). There currently exists a technical limitation preventing us from being able to display them in a list of Microsoft Entra group members. This limitation isn't true for Azure DevOps groups. That being said, a service principal still inherits any group permissions set on top of a Microsoft Entra group they belong to.
- Users in a Microsoft Entra group aren't part of an Azure DevOps organization immediately just because an admin creates a group and adds a Microsoft Entra group to it. We have a process called "materialization" that happens once a user from a Microsoft Entra group signs in to the organization for the first time. A user signing into an organization allows us to determine which users should be granted a license. Since sign-in isn't possible for service principals, an admin must explicitly add it to an organization as described earlier.
- You can't modify a service principal's display name or avatar on Azure DevOps.
- Service principals get licenses in each organization they get added to, even if multi-organization billing is selected.

5. Get a Microsoft Entra ID token

(a) Acquire a Microsoft Entra ID token programmatically

Acquiring an access token for a managed identity can be done by following along with the Microsoft Entra ID documentation. See the examples for service principals and managed identities.

The returned access token is a JSON web token (JWT) with the defined roles, which can be used to access organization resources using the token as *Bearer*.

(b) Acquire a Microsoft Entra ID token with the Azure CLI

For ad-hoc operations, it might be easier to acquire a one-off Microsoft Entra ID token through the Azure CLI. This approach is preferred for operations that don't need a persistent token to be regularly rotated, like API calls or git clone operations.

Prerequisites

- **Azure tenant id and subscription id**: Make sure the subscription is associated with the tenant connected to the Azure DevOps organization you're trying to access. If you don't know your tenant or subscription ID, you can find it in the Azure portal.
- **Azure app client ID and client secret**
- **Azure CLI**

These instructions are provided by the Databricks docs and more details can be found on their page.

1. Sign in to the Azure CLI as the service principal using the `az devops login` command.
2. Follow the on-screen instructions and finish signing in.

PowerShell

```
# To authenticate a service principal with a password or cert:
az login --service-principal -u <app-id> -p <password-or-cert> --tenant
<tenant>

# To authenticate a managed identity:
az login --identity
```

3. Set the right correct subscription for the signed-in service principal by entering the command:

PowerShell

```
az account set -s <subscription-id>
```

4. Generate a Microsoft Entra ID access token with the `az account get-access-token` the Azure DevOps resource ID: `499b84ac-1321-427f-aa17-267ca6975798`.

PowerShell

```
$accessToken = az account get-access-token --resource 499b84ac-1321-427f-
aa17-267ca6975798 --query "accessToken" --output tsv
```

5. Now, you can use `az cli` commands per usual. Let's try to call an Azure DevOps API by passing it in the headers as a `Bearer` token:

PowerShell

```
$apiVersion = "7.1-preview.1"
$uri = "https://dev.azure.com/${yourOrgname}/_apis/projects?api-
version=${apiVersion}"
```

```
$headers = @{
    Accept = "application/json"
    Authorization = "Bearer $accessToken"
}
Invoke-RestMethod -Uri $uri -Headers $headers -Method Get | Select-Object -
ExpandProperty value ` | Select-Object id, name
```

> **Note**
>
> Use the Azure DevOps application ID, not our resource URI, for generating tokens.

6. Use the Microsoft Entra ID token to authenticate to Azure DevOps resources

In the following video example, we move from authenticating with a PAT to using a token from a service principal. We start by using a client secret for authentication, then move to using a client certificate.
https://learn-video.azurefd.net/vod/player?id=8397f848-b946-4c30-8d0c-78aa4784a549&locale=en-us&embedUrl=%2Fazure%2Fdevops%2Fintegrate%2Fget-started%2Fauthentication%2Fservice-principal-managed-identity

Another example demonstrates how to connect to Azure DevOps using a User Assigned Managed Identity within an Azure Function.
https://learn-video.azurefd.net/vod/player?id=f4a20e00-c2df-49cc-9b86-7454bd84b9ce&locale=en-us&embedUrl=%2Fazure%2Fdevops%2Fintegrate%2Fget-started%2Fauthentication%2Fservice-principal-managed-identity

Follow along with these examples by finding the app code in our collection of sample apps.

Some common scenarios for authenticating with service principals besides making Azure DevOps REST API calls can be found in these docs:

- Connect your service principal to a NuGet feed with Nuget.exe or dotnet.
- Publish extensions to the Visual Studio Marketplace via command line with your service principal.
- Create secret-free service connections in Azure Pipelines backed by service principals or managed identities.
- Clone repos using a service principal with Git Credential Manager

How service principals differ from users

- You can't modify a service principal's display name or avatar on Azure DevOps.
- A service principal counts as a license for each organization it joins, even with multi-organization billing.
- Service principals can't be organization owners or create organizations.
- Service principals can't create tokens like personal access tokens (PATs) or SSH Keys. They can generate their own Microsoft Entra ID tokens to call Azure DevOps REST APIs.
- Service principals don't support Azure DevOps OAuth.

FAQs

Q: Why should I use a service principal or a managed identity instead of a PAT?

A: Many of our customers seek out a service principal or managed identity to replace an existing PAT (personal access token). Such PATs often belong to a service account (shared team account) that is using them to authenticate an application with Azure DevOps resources. PATs must be laboriously rotated every so often (minimum 180 days). Improperly stored PATs can fall into the wrong hands and last the duration of its often longer lifespan. Microsoft Entra tokens expire every hour, limiting the overall risk factor when leaked. For common PAT scenarios, we share some examples on how you might explore using a Microsoft Entra token instead.

You can't use a service principal to create a personal access token.

Q: What are the rate limits on service principals and managed identities?

A: Service principals and managed identities have the same rate limits as users.

Q: Will using this feature cost more?

A: Service principals and managed identities are priced similarly as users, based on the access level. One notable change pertains to how we treat "multi-org billing" for service principals. Users get counted as only one license, no matter how many organizations they're in. Service principals get counted as one license per each organization the user's in. This scenario is similar to standard "user assignment-based billing."

Q: Can I add a managed identity from a different tenant to my organization?

A: You can only add a managed identity from the same tenant that your organization is connected to. However, we have a workaround that allows you to set up a managed identity in the "resource tenant," where are all of your resources are. Then, you can enable it to be used by a service principal in the "target tenant," where your organization is connected. Do the following steps as a workaround:

1. Create a user-assigned managed identity in Azure portal for your resource tenant.
2. Connect it to a virtual machine and assign this managed identity to it.
3. Create a key vault and generate a certificate (can't be of type "PEM"). When you generate this certificate, a secret with the same name is also generated, which we use later.
4. Grant access to the managed identity so that it can read the private key from the key vault. Create an access policy in the key vault with the "Get/List" permissions (under "Secret permissions" and search for the managed identity under "Select principal."
5. Download the created certificate in "CER" format, which ensures that it doesn't contain the private part of your certificate.
6. Create a new application registration in the target tenant.
7. Upload the downloaded certificate to this new application in the "Certificates & secrets" tab.
8. Add this application's service principal to the Azure DevOps organization we want it to access and remember to set up the service principal with any required permissions.
9. Get a Microsoft Entra access token from this service principal that makes use of the managed identity certificate with this code sample:

> **Note**
>
> Always regularly rotate your certificates.

cs

```cs
public static async Task<string> GetSecret(string keyVaultName, string
secretName)
{
    var keyVaultUri = new Uri("https://" + keyVaultName +
".vault.azure.net");
    var client = new SecretClient(keyVaultUri, new
ManagedIdentityCredential());
    var keyVaultSecret = await client.GetSecretAsync(secretName);
```

```
    var secret = keyVaultSecret.Value;
    return secret.Value;
}

private static async Task<AuthenticationResult>
GetAppRegistrationAADAccessToken(string applicationClientID, string
appTenantId)
{
    IConfidentialClientApplication app;

    byte[] privateKeyBytes =
Convert.FromBase64String(GetSecret(keyVaultName, secretName));
    X509Certificate2 certificateWithPrivateKey = new
X509Certificate2(privateKeyBytes, (string)null,
X509KeyStorageFlags.MachineKeySet);

    app = ConfidentialClientApplicationBuilder.Create(applicationClientID)
        .WithCertificate(certificateWithPrivateKey)
        .WithAuthority(new Uri(string.Format(CultureInfo.InvariantCulture,
"https://login.microsoftonline.com/{0}", appTenantId)))
        .Build();
    app.AddInMemoryTokenCache();

    string AdoAppClientID = "499b84ac-1321-427f-aa17-267ca6975798/.default";
    string[] scopes = new string[] { AdoAppClientID };

    var result = await app.AcquireTokenForClient(scopes).ExecuteAsync();

    return result;
}
```

Potential errors

The Git repository with name or identifier '{`repoName`}' doesn't exist or you don't have permissions for the operation you're attempting.

Ensure that the service principal has at least a "Basic" license to access repositories. A "Stakeholder" license isn't sufficient.

Failed to create service principal with object ID '{`provided objectId`}'

There's no service principal with the `provided objectId` in the tenant connected to your organization. One common reason is that you're passing in the object ID of the app

registration, instead of the object ID of its service principal. Remember, a service principal is an object that represents the application for a given tenant, it's not the application itself. The `service principal object ID` can be found in your tenant's "Enterprise Applications" page. Search for the application's name and select on the "Enterprise Application" result that returns. This result is the page of the service principal / enterprise application and you can use the Object ID found on this page to create a service principal in Azure DevOps.

Access Denied: {`ID of the caller identity`} needs the following permissions on the resource Users to perform this action: Add Users

This error might be due to one of the following reasons:

- You're not the owner of the organization, project collection administrator, or a project or team administrator.
- You're a project or team administrator, but the policy 'Allow team and project administrators to invite new users' is disabled.
- You're a project or team administrator who can invite new users, but you're trying to assign a license when you invite a new user. Project or team administrators aren't allowed to assign a license to new users. Any new invited user is added at the default access level for new users. Contact a PCA to change the license access level.

Azure DevOps Graph List API returns an empty list, even though we know there are service principals in the organization

The Azure DevOps Graph List API might return an empty list, even if there are still more pages of users to return. Use the `continuationToken` to iterate through the lists, and you can eventually find a page where the service principals are returned. If a `continuationToken` is returned, that means there are more results available through the API. While we have plans to improve upon this logic, at this moment, it's possible that the first X results return empty.

TF401444: Sign-in at least once as {`tenantId`tenantId`\`servicePrincipalObjectId`} in a web browser to enable access to the service.

If the service principal wasn't invited to the organization, you might come across the following error. Ensure that the service principal is added to the appropriate organization and has all permissions needed to access any required resources.

Related articles

- Explore sample apps
- Review the Service Principals Entitlements API reference
- Review the Service Principal Graph API reference

Feedback

Was this page helpful? Yes No

Provide product feedback

Use personal access tokens

Article • 01/08/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

A Personal Access Token (PAT) serves as an alternative password for authenticating into Azure DevOps. This PAT identifies you and determines your accessibility and scope of access. Therefore, treat PATs with the same level of caution as passwords.

> **ⓘ Important**
>
> We recommend using **Microsoft Entra tokens**. For more information about our efforts to reduce PAT usage, see **our blog**. Review our **authentication guidance** to choose the appropriate authentication mechanism for your needs.

When you use Microsoft tools, your Microsoft account (MSA) or Microsoft Entra ID is recognized and supported. If you use tools that don't support Microsoft Entra accounts or don't want to share your primary credentials, PATs can be a suitable alternative. However, we recommend using Microsoft Entra tokens over PATs whenever possible.

You can manage PATs through the following methods:

- **User interface (UI):** Through user settings, as detailed in this article.
- **PAT Lifecycle Management APIs**
- Git credential managers for Git operations. Credential managers facilitate token management. Without one, users must input their credentials each time.

Prerequisites

- **Permissions:**
  - Have permission to access and modify your user settings where PATs are managed.
    - **Check permissions:** To check your permissions, do either of the following processes in Azure DevOps:
      - Go to **your profile** and select **User settings** > **Personal access tokens**. If you can see and manage your PATs here, you have the necessary permissions.
      - Go to your project and select **Project settings** > **Permissions**. Find your user account in the list and check the permissions assigned to you. Look for permissions related to managing tokens or user settings.

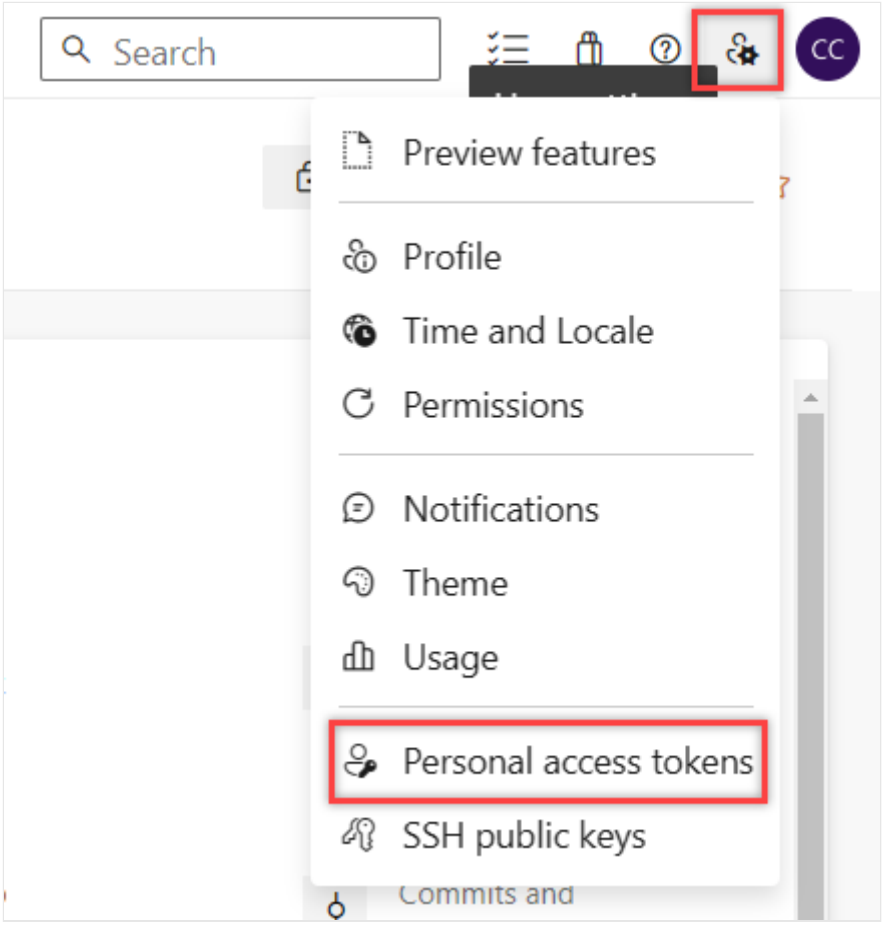
- If your organization has policies in place, an Azure DevOps Administrator might need to grant you specific permissions or add you to an allowlist to create and manage PATs.
- PATs are connected to the user account that minted the token. Depending on the tasks the PAT performs, you might need more permissions yourself.

- **Access levels:** Have at least Basic access.
- **Security best practices:** Familiarize yourself with security best practices for managing PATs. **Use them only when necessary and always regularly rotate them.**

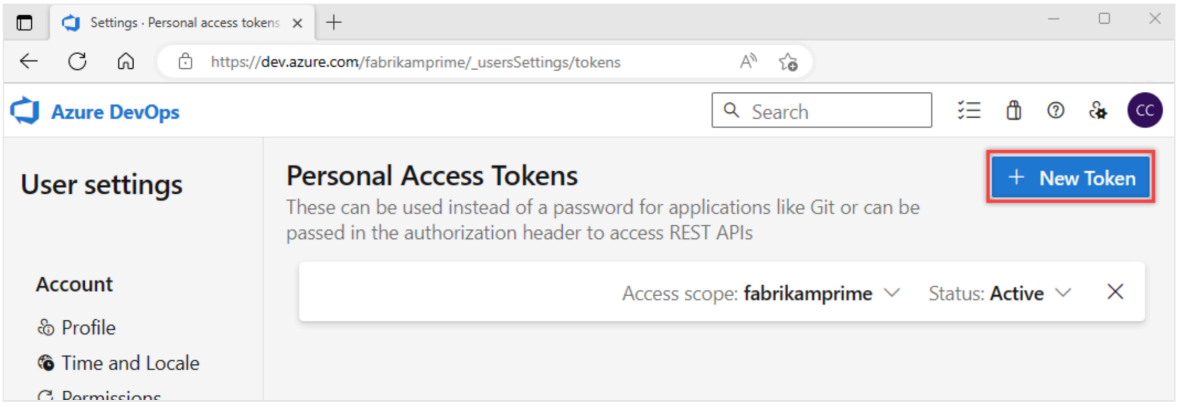
Create a PAT

1. Sign in to your organization (`https://dev.azure.com/{Your_Organization}`).

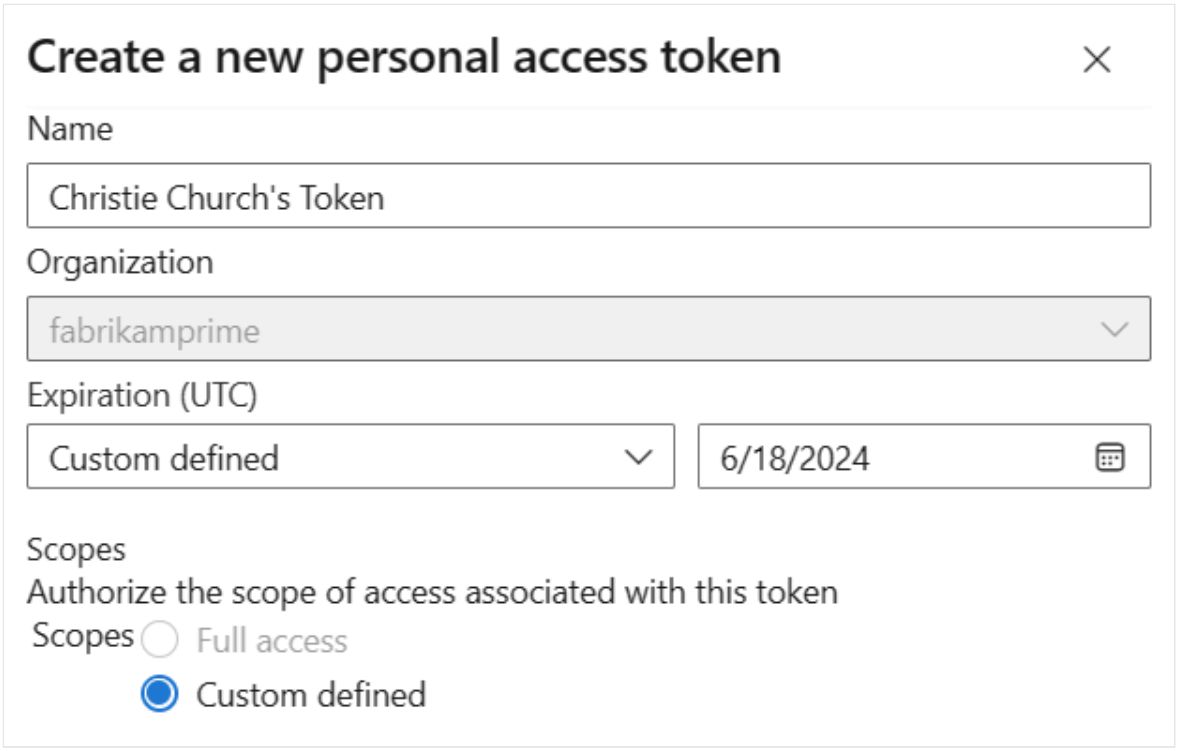
2. From your home page, open user settings and select **Personal access tokens**.



3. Select + **New Token**.

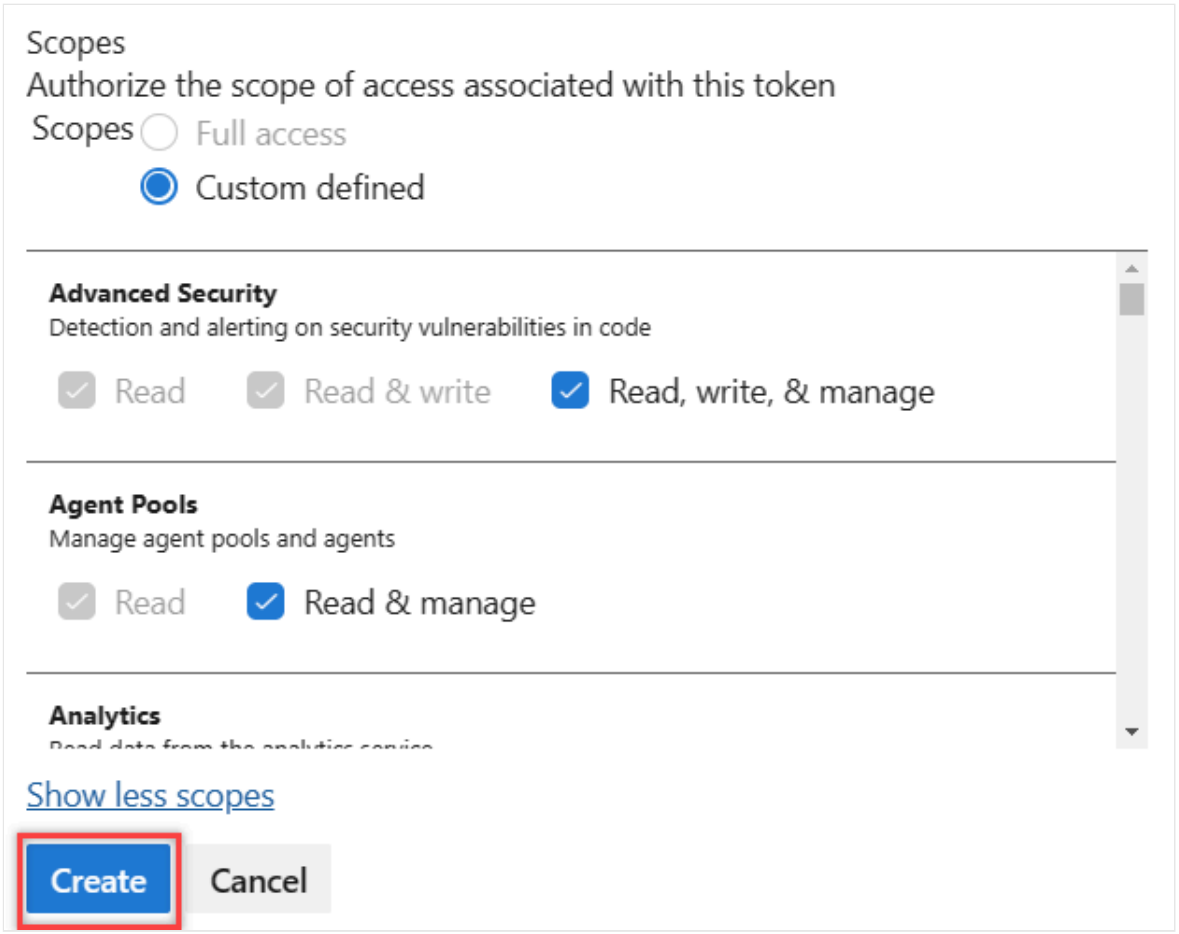


4. Name your token, select the organization where you want to use the token, and then set your token to automatically expire after a set number of days.



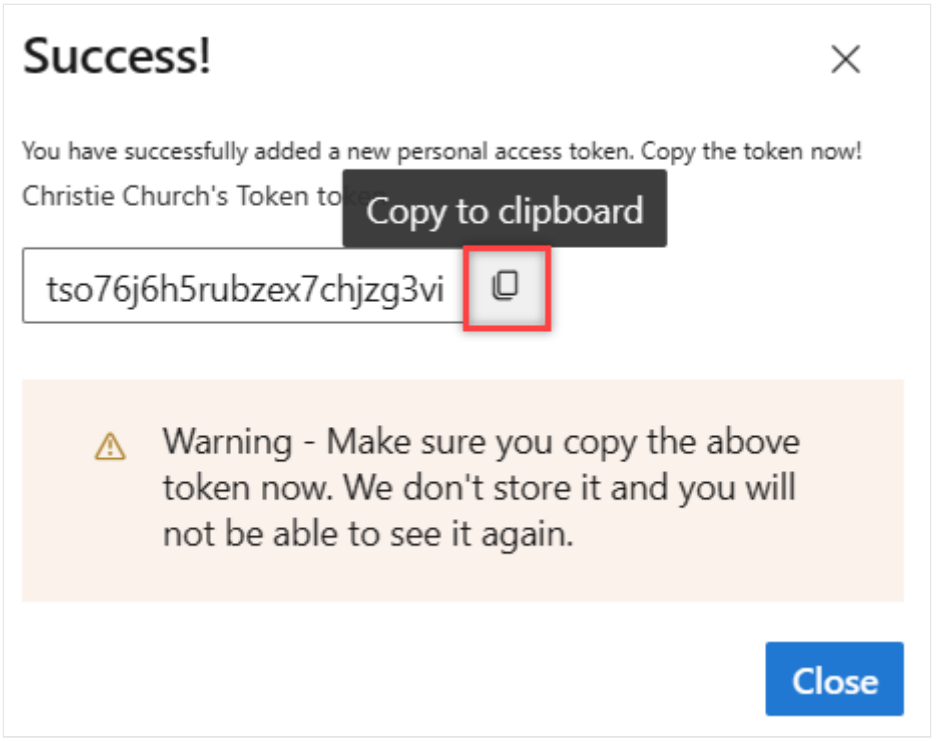
5. Select the scopes for this token to authorize for *your specific tasks*.

   For example, to create a token for a build and release agent to authenticate to Azure DevOps, set the token's scope to **Agent Pools (Read & manage)**. To read audit log events and manage or delete streams, select **Read Audit Log**, and then select **Create**.



> **Note**
>
> You might be restricted from creating full-scoped PATs. If so, your Azure DevOps Administrator in Microsoft Entra ID enabled a policy that limits you to a specific custom-defined set of scopes. For more information, see Manage PATs with policies/Restrict creation of full-scoped PATs. For a custom-defined PAT, the required scope for accessing the Component Governance API, `vso.governance`, isn't selectable in the UI.

6. When you're done, copy the token and store it in a secure location. For your security, it doesn't display again.



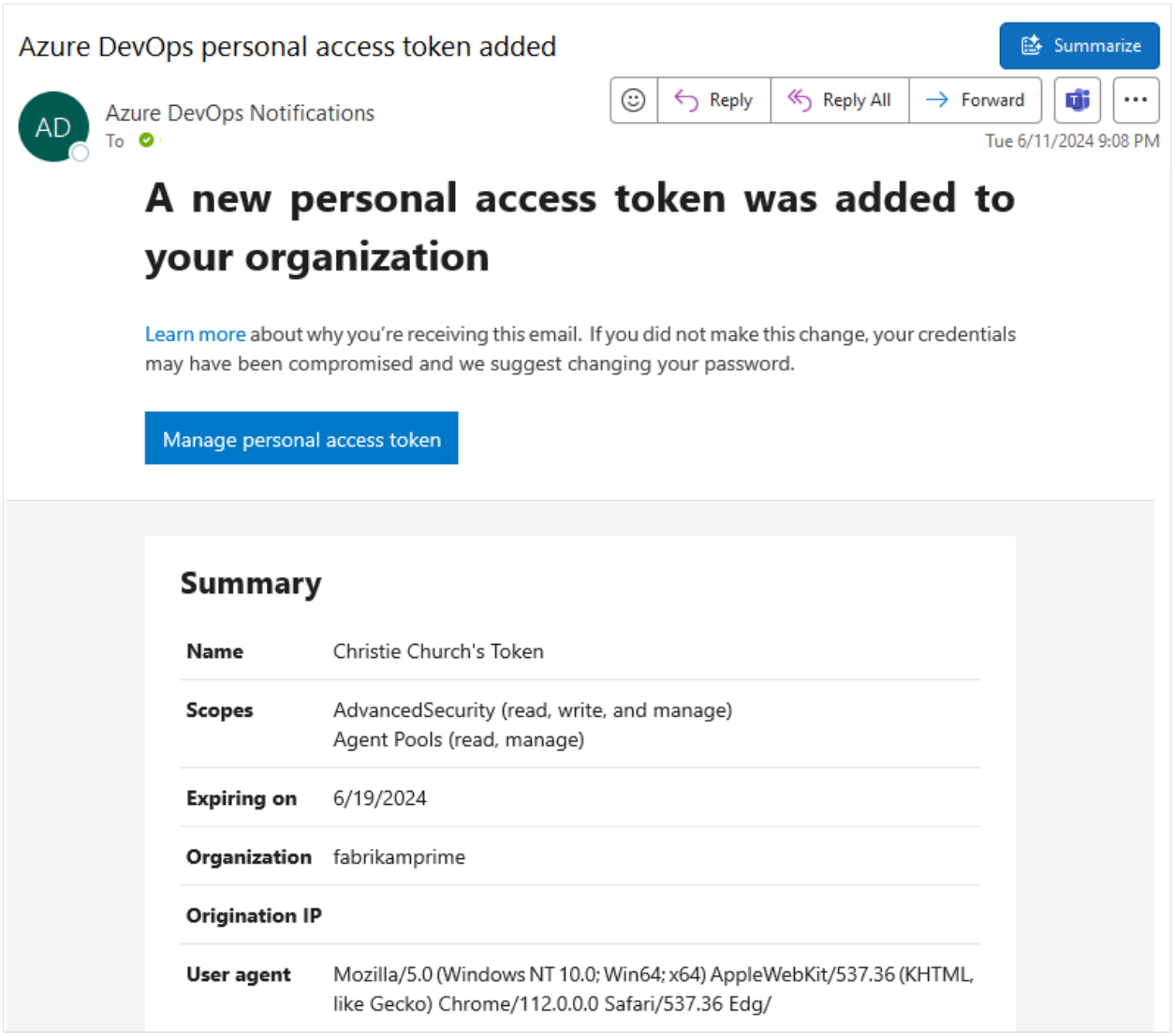
Use your PAT anywhere your user credentials are required for authentication in Azure DevOps.

> **Important**
>
> - Treat a PAT with the same caution as your password and keep it confidential.
> - Sign in with your new PAT within 90 days for organizations backed by Microsoft Entra ID; otherwise, the PAT becomes inactive. For more information, see User sign-in frequency for Conditional Access.

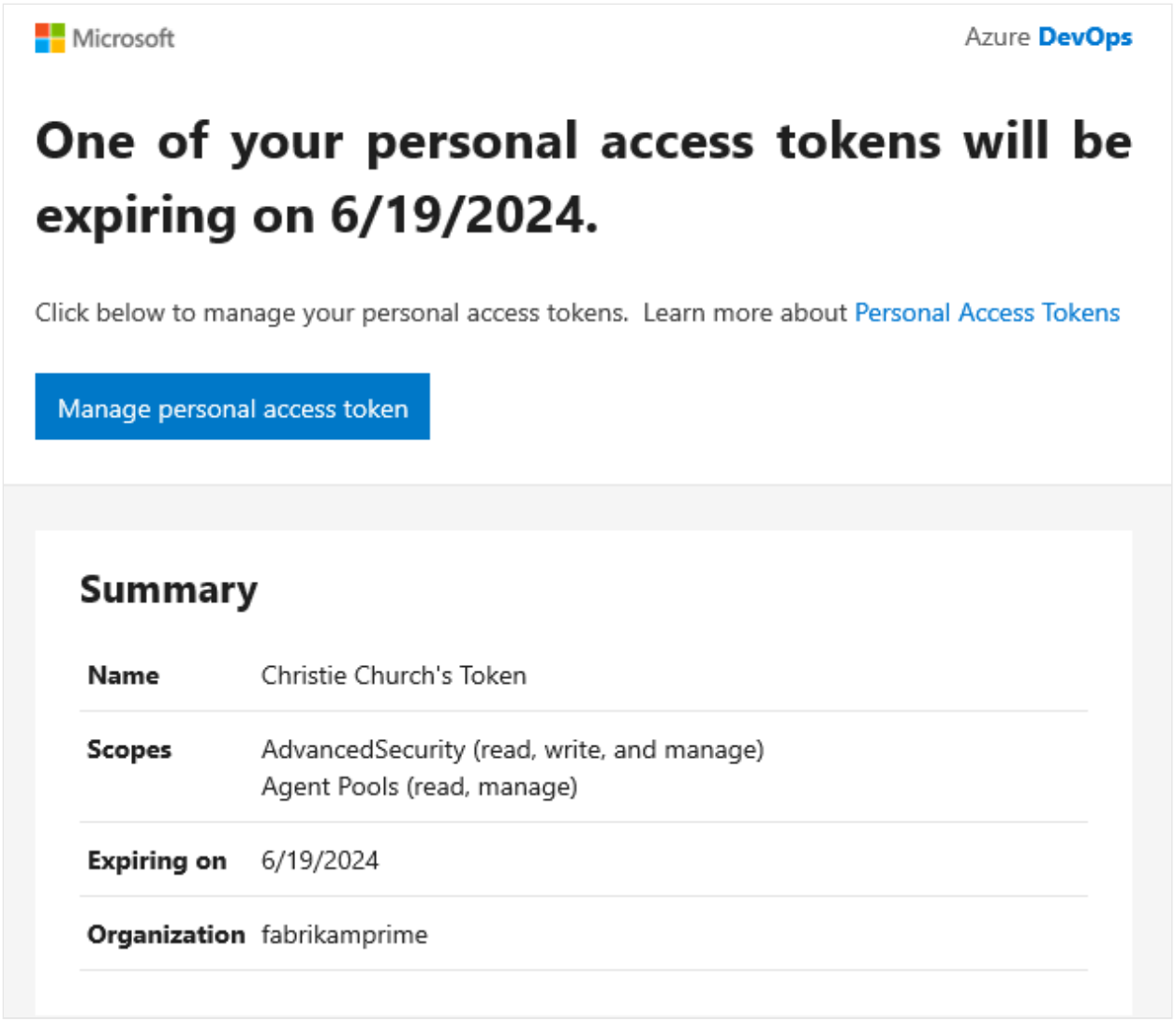
Notifications

During the lifespan of a PAT, users receive two notifications: the first at the time of creation and the second seven days before its expiration.

After you create a PAT, you receive a notification similar to the following example. This notification serves as confirmation that your PAT was successfully added to your organization.



The following image shows an example of the seven-day notification before your PAT expires.



Unexpected notification

If you receive an unexpected PAT notification, it might mean that an administrator or tool created a PAT for you. Here are some examples:

- A token named "git: `https://dev.azure.com/{Your_Organization}` on YourMachine" gets created when you connect to an Azure DevOps Git repo via git.exe.
- A token named "Service Hooks: Azure App Service: Deploy web app" gets created when you or an administrator sets up an Azure App Service web app deployment.
- A token named "WebAppLoadTestCDIntToken" gets created when web load testing gets set up as part of a pipeline by you or an administrator.
- A token named "Microsoft Teams Integration" gets created when a Microsoft Teams Integration Messaging Extension gets set up.

> ⚠ **Warning**
>
> - Revoke the PAT (and change your password) if you suspect it exists in error.

- Check with your administrator if you're a Microsoft Entra user to see if an unknown source or location accessed your organization.
- Review the FAQ on accidental PAT check-ins to public GitHub repositories.

Use a PAT

Your PAT serves as your digital identity, much like a password. You can use PATs as a quick way to do one-off requests or prototype an application locally.

> **ⓘ Important**
>
> When your code is working, it's a good time to switch from basic auth to Microsoft Entra OAuth. You can use Microsoft Entra ID tokens anywhere a PAT gets used, unless specified further in this article.

You can use a PAT in your code to authenticate REST APIs requests and automate workflows. To do so, include the PAT in the authorization header of your HTTP requests.

Windows

To provide the PAT through an HTTP header, first convert it to a `Base64` string. The following example shows how to convert to `Base64` using C#.

cs

```
Authorization: Basic BASE64_USERNAME_PAT_STRING
```

The resulting string can then be provided as an HTTP header in the following format.

The following sample uses the HttpClient class in C#.

cs

```
public static async void GetBuilds()
{
    try
    {
        var personalaccesstoken = "PATFROMWEB";

        using (HttpClient client = new HttpClient())
```

```
        {
            client.DefaultRequestHeaders.Accept.Add(
                new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/jso
n"));

            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic",
                Convert.ToBase64String(
                    System.Text.ASCIIEncoding.ASCII.GetBytes(
                        string.Format("{0}:{1}", "",
personalaccesstoken))));

            using (HttpResponseMessage response = client.GetAsync(

"https://dev.azure.com/{organization}/{project}/_apis/build/builds?api-
version=5.0").Result)
            {
                response.EnsureSuccessStatusCode();
                string responseBody = await
response.Content.ReadAsStringAsync();
                Console.WriteLine(responseBody);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Tip

When you're using variables, add a `$` at the beginning of the string, like in the following example.

cs

```
public static async void GetBuilds()
{
   try
  {
      var personalaccesstoken = "PATFROMWEB";

      using (HttpClient client = new HttpClient())
       {
           client.DefaultRequestHeaders.Accept.Add(
              new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("applicatio
n/json"));

           client.DefaultRequestHeaders.Authorization = new
```

```
AuthenticationHeaderValue("Basic",
                Convert.ToBase64String(
                    System.Text.ASCIIEncoding.ASCII.GetBytes(
                        string.Format("{0}:{1}", "",
personalaccesstoken))));

          using (HttpResponseMessage response = client.GetAsync(

$"https://dev.azure.com/{organization}/{project}/_apis/build/builds
?api-version=5.0").Result)
            {
                response.EnsureSuccessStatusCode();
                string responseBody = await
response.Content.ReadAsStringAsync();
                Console.WriteLine(responseBody);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Some more examples of how to use PATs can be found in the following articles:

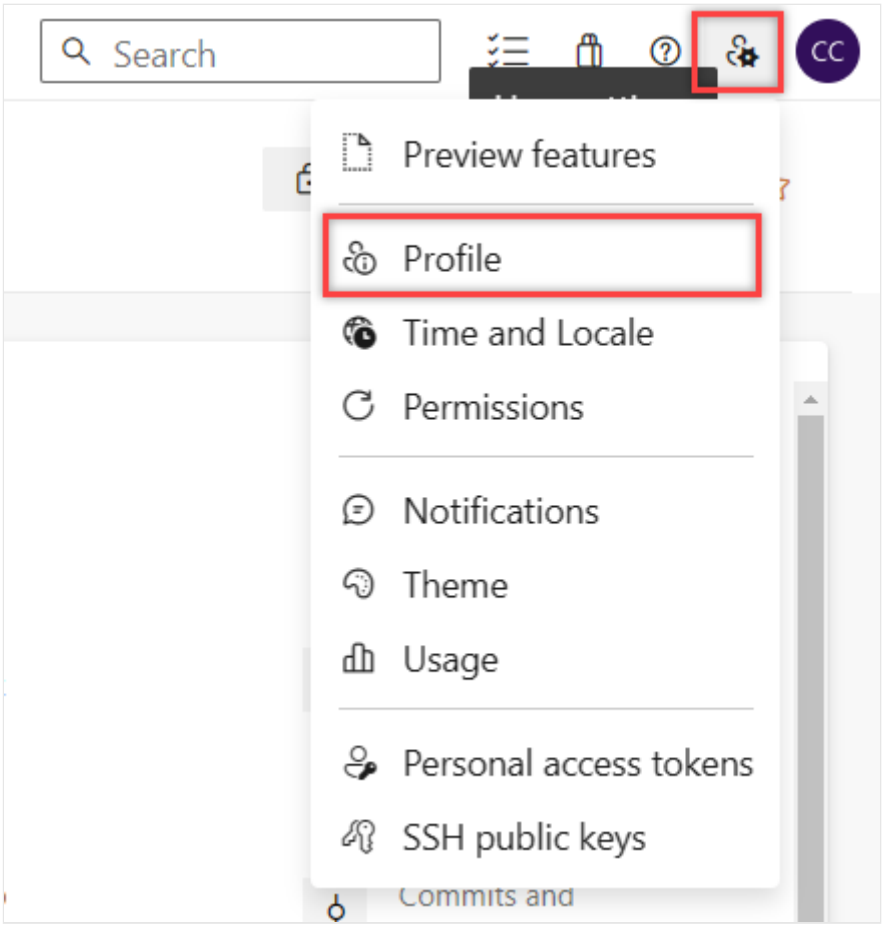
- Authenticate with your Git repos
- Set up Git credential managers (GCM) to connect with Git repositories
- Use NuGet on a Mac
- Authenticate reporting clients
- Get started with Azure DevOps CLI

Modify a PAT

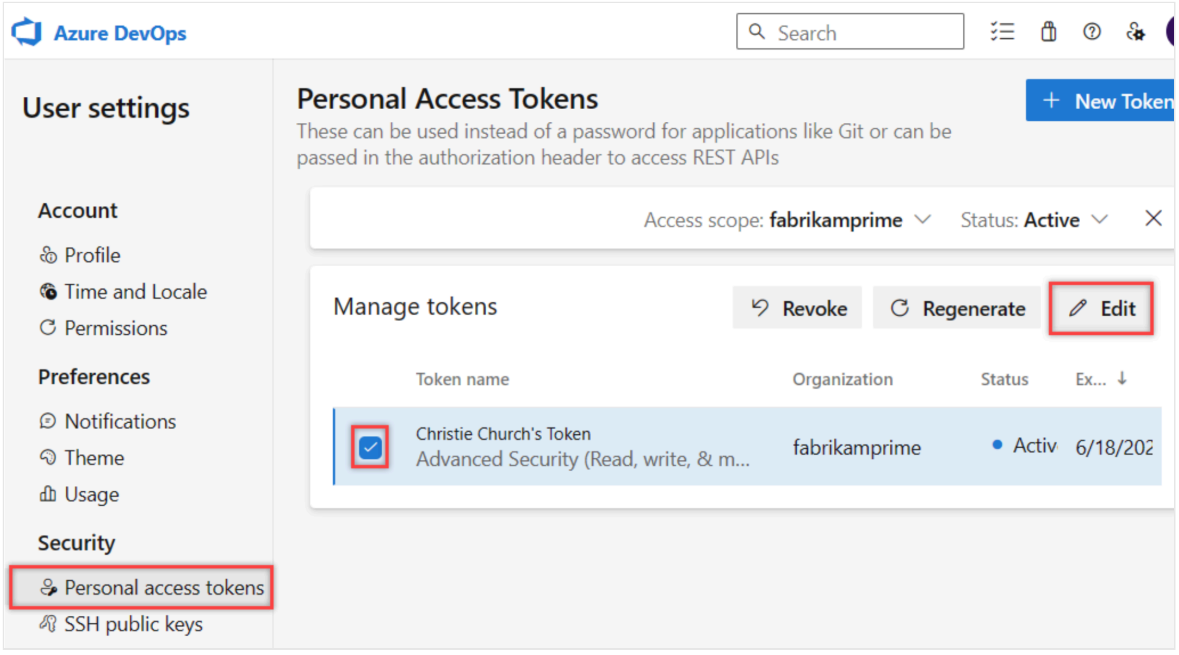
Do the following steps to:

- Regenerate a PAT to create a new token, which invalidates the previous one.
- Extend a PAT to increase its validity period.
- Alter the scope of a PAT to change its permissions.

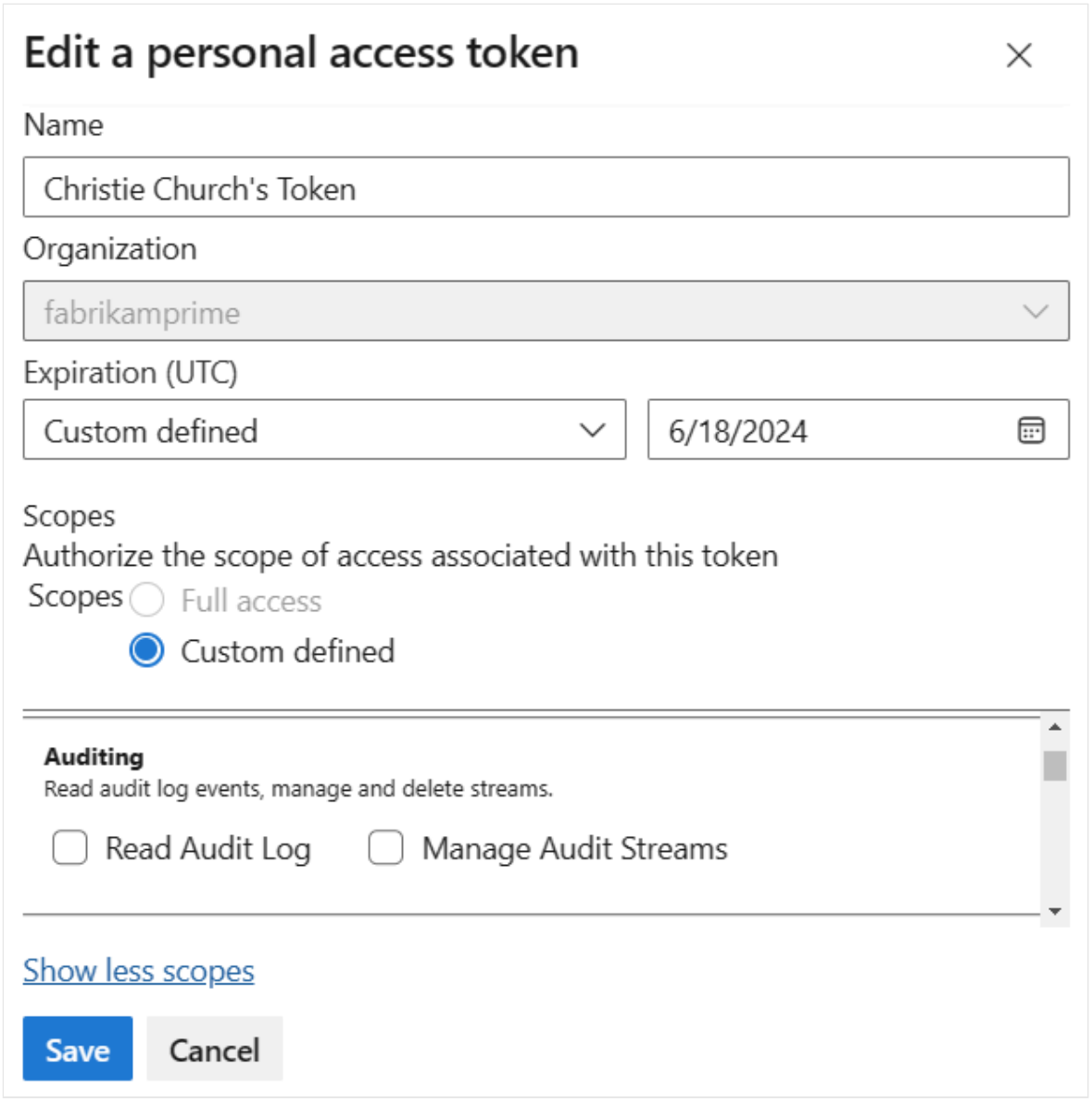
1. From your home page, open your user settings, and then select **Profile**.



2. Under Security, select **Personal access tokens**. Select the token you want to modify, and then **Edit**.



3. Edit the token name, token expiration, or the scope of access associated with the token, and then select **Save**.

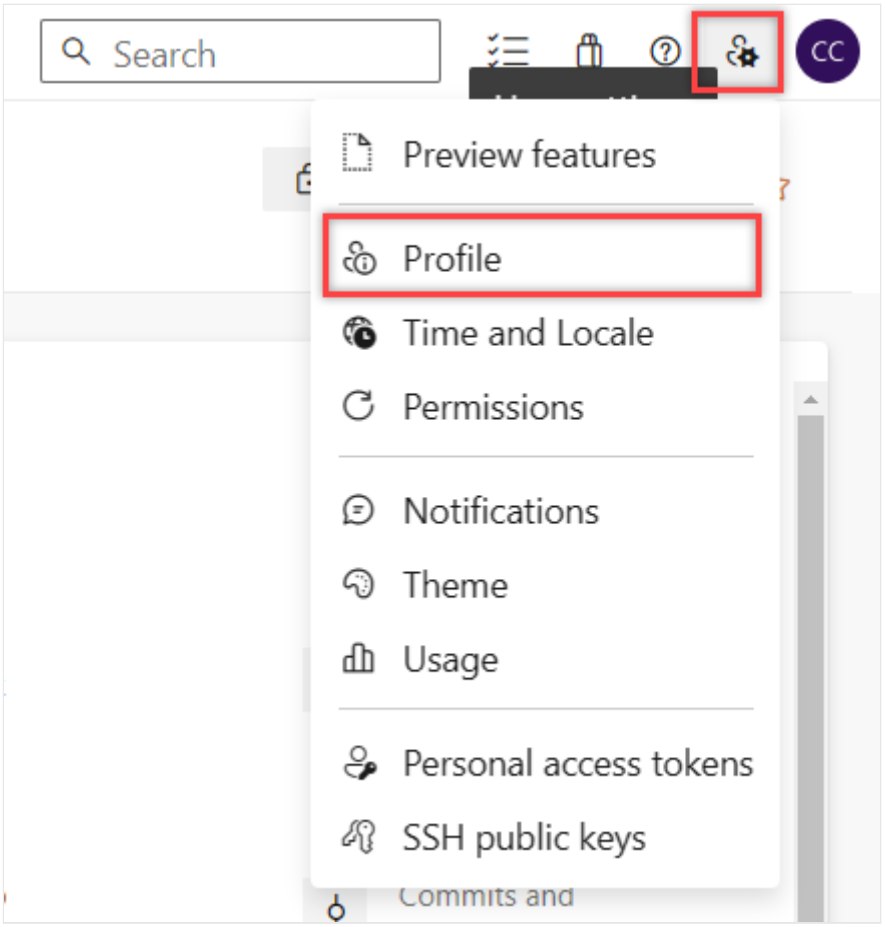


Revoke a PAT

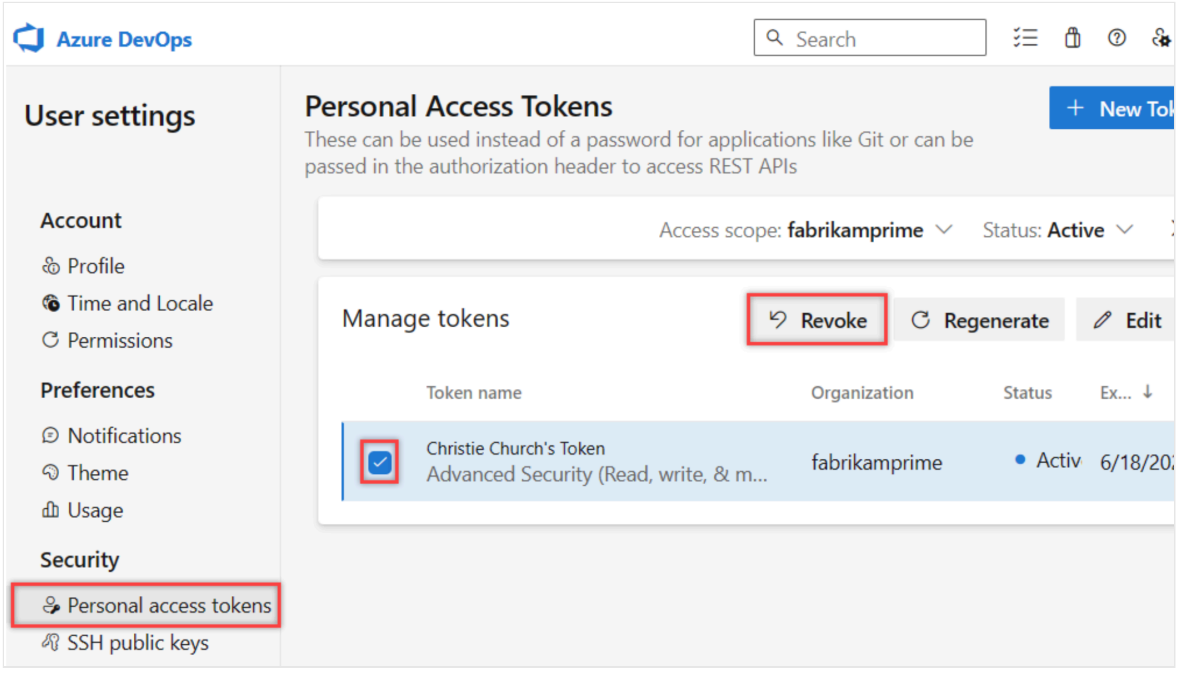
You can revoke a PAT at any time for these and other reasons:

- Revoke a PAT if you suspect it's compromised.
- Revoke a PAT when it's no longer needed.
- Revoke a PAT to enforce security policies or compliance requirements.

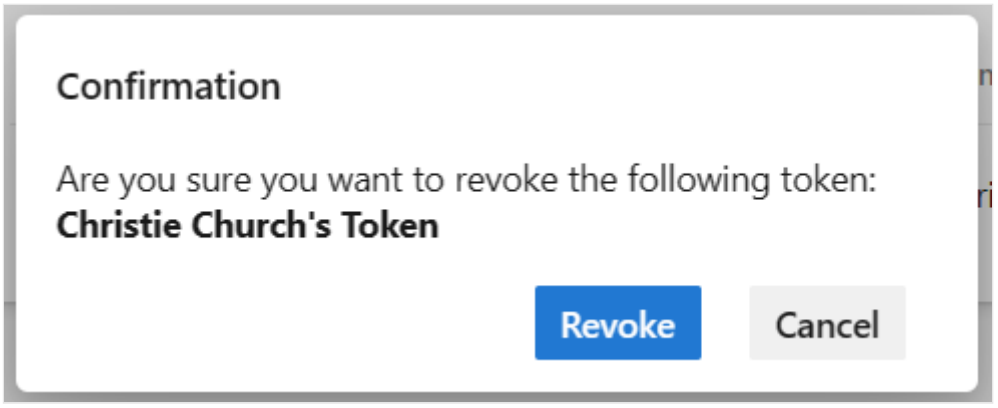
1. From your home page, open your user settings, and then select **Profile**.



2. Under Security, select **Personal access tokens**. Select the token for which you want to revoke access, and then select **Revoke**.



3. Select **Revoke** in the confirmation dialog.



For more information, see Revoke user PATs for admins.

Changes to format

As of July 2024, we significantly changed the format of PATs issued by Azure DevOps. These changes provide more security benefits and improve secret detection tooling available through our leaked PAT detection tooling or partner offerings. This new PAT format follows the recommended format across all Microsoft products. The inclusion of more identifiable bits improves the false positive detection rate of these secret detection tools and enables us to mitigate detected leaks faster.

Key changes:

- **Increased token length:** The new tokens are now **84** characters long, with 52 characters being randomized data. This increased length improves overall entropy, making the tokens more resistant to potential brute force attacks.
- **Fixed signature:** Tokens issued by our service include a fixed `AZDO` signature at positions 76-80.

Action required:

- **Regenerate existing PATs:** We strongly recommend regenerating all PATs currently in use to take advantage of these security enhancements.
- **Integrator support:** Integrators should update their systems to accommodate both the new and existing token lengths.

> **ⓘ Important**
>
> Both formats remain valid for the foreseeable future, but we **actively encourage customers to transition to the new 84-character format**. As adoption of the new format increases, we consider retiring the older 52-character format and all tokens issued in that style.

Best practices for using PATs

Consider alternatives

- Acquire a Microsoft Entra token via Azure CLI for ad-hoc requests that lasts for one hour instead of minting a longer-lived PAT.
- Use credential managers like Git Credential Manager or Azure Artifacts Credential Manager for credential management simplification. These tools might offer options to use Microsoft Entra tokens as the default authentication instead of PATs.

Creating PATs

- Avoid putting personal data in the PAT name. Don't rename the PAT token string as the name of your token.
- Select only the organization your PAT needs to access if it doesn't need to access multiple organizations. For workflows requiring access to multiple organizations, create a separate global PAT for that workflow.
- Choose only the necessary scopes for each PAT. If possible, create multiple PATs for each workflow with fewer scopes instead of a single fully scoped PAT. If your PAT only needs read permissions, don't provide it write permissions until necessary.
- Keep PAT lifespans short (weekly is ideal, even shorter is better) and regularly rotate or regenerate them via the UI or the PAT Lifecycle Management APIs.

Managing PATs

- **Always** store your PATs in a secure key management solution like Azure KeyVault.
- Revoke PATs when they're no longer needed. Tenant admins can revoke PATs for their organization users if the PAT is compromised.
- Rotate your PATs to use the new PAT format for better leaked secret detection and revocation by our first-party tools.

For admins

Tenant admins can set policies to restrict global PAT creation, full scoped PAT creation, and long-lived PAT duration. They can also enable policies to automatically revoke leaked PATs detected in public repositories. Use these policies to improve your company's security.

FAQs

Q: Why can't I edit or regenerate a PAT scoped to a single organization?

A: Sign into the organization where your PAT is scoped. You can view all your PATs while signed into any organization in the same Microsoft Entra ID, but you can only edit organization-scoped tokens when signed into the specific organization.

Q: What happens to a PAT if a user account is disabled?

A: When a user is removed from Azure DevOps, the PAT invalidates within 1 hour. If your organization is connected to Microsoft Entra ID, the PAT also invalidates in Microsoft Entra ID, as it belongs to the user. We recommend rotating the PAT to another user or service account to keep services running.

Q: Is there a way to renew a PAT via REST API?

A: Yes, you can renew, manage, and create PATs using our PAT Lifecycle Management APIs.

Q: Can I use PATs with all Azure DevOps REST APIs?

A: No. You can use basic auth with most Azure DevOps REST APIs, but organizations and profiles and the PAT Management Lifecycle APIs only support Microsoft Entra OAuth. For an example on how to set up a Microsoft Entra app to call such APIs, see Manage PATs using the REST API.

Q: What happens if I accidentally check my PAT into a public repository on GitHub?

A: Azure DevOps scans for PATs checked into public repositories on GitHub. When we find a leaked token, we immediately send a detailed email notification to the token owner and log an event in your Azure DevOps organization's audit log. Unless you disabled the *Automatically revoke leaked personal access tokens* policy, we immediately revoke the leaked PAT. We encourage affected users to mitigate the issue by revoking the leaked token and replacing it with a new token. For more information, see Revoke leaked PATs automatically.

Q: Can I use a personal access token as an ApiKey to publish NuGet packages to an Azure Artifacts feed using the dotnet/nuget.exe command line?

A: No. Azure Artifacts doesn't support passing a PAT as an ApiKey. When using a local development environment, we recommend installing the Azure Artifacts Credential Provider to authenticate with Azure Artifacts. For more information, see the following examples: dotnet, NuGet.exe. If you want to publish your packages using Azure Pipelines, use the NuGet Authenticate task to authenticate with your feed. See example.

Q: Why did my PAT stop working?

A: PAT authentication requires you to regularly sign into Azure DevOps using the full authentication flow. Signing in once every 30 days is sufficient for many users, but you might need to sign in more frequently depending on your Microsoft Entra configuration. If your PAT stops working, first try signing into your organization and complete the full authentication prompt. If your PAT still doesn't work, check if it expired.

Q: How do I create access keys that aren't tied to a specific person for deployment purposes?

A: In Azure DevOps, you can create access keys that aren't tied to a specific person by using Service Principals or Managed Identities. For more information, see Manage service connections and Use Azure Key Vault secrets in Azure Pipelines.

Related articles

- Use policies to manage personal access tokens for users
- Revoke user PATs (for admins)
- Learn about security, authentication, and authorization
- Review default permissions and access for Azure DevOps
- Manage personal access tokens (PATs) using REST API
- Manage service principals and managed identities in Azure DevOps

ⓘ **Note:** The author created this article with assistance from AI. Learn more

---

Feedback

Was this page helpful? Yes No

Provide product feedback

Authorize access to REST APIs with OAuth 2.0

Article • 10/29/2024

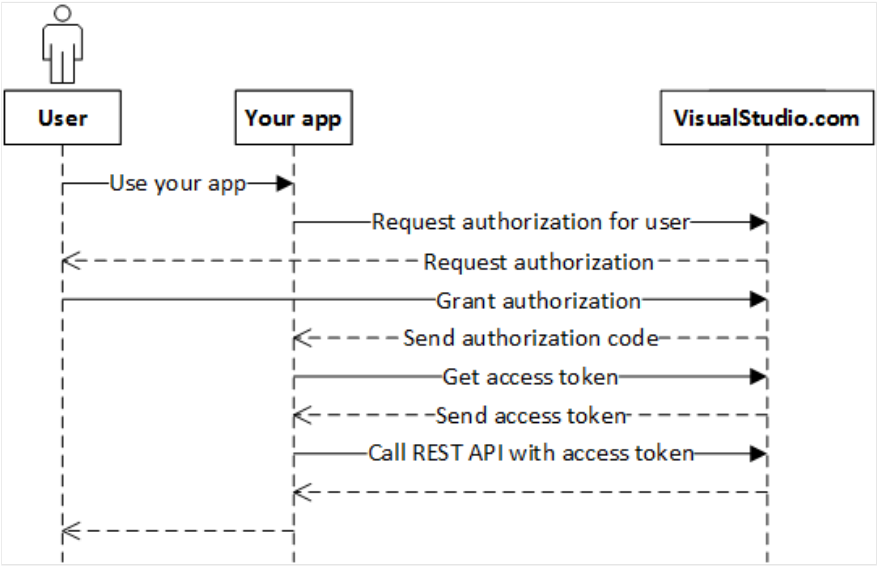
Azure DevOps Services

Learn how to authenticate your web app users for REST API access, so your app doesn't continue to ask for usernames and passwords.

> **Note**
>
> - The following guidance is intended for Azure DevOps Services users since OAuth 2.0 isn't supported on Azure DevOps Server. Client Libraries are a series of packages built specifically for extending Azure DevOps Server functionality. For on-premises users, we recommend using Client Libraries, Windows Auth, or personal access tokens (PATs) to authenticate on behalf of a user.
> - For more information, see the C# OAuth GitHub sample.

About OAuth 2.0

Azure DevOps Services uses the OAuth 2.0 protocol to authorize your app for a user and generate an access token. Use this token when you call the REST APIs from your application. When you call Azure DevOps Services APIs for that user, use that user's access token. Access tokens expire, so refresh the access token if expired.



Available OAuth models

> **Important**
>
> When creating a new OAuth 2.0 app, use Microsoft Entra ID OAuth. Azure DevOps OAuth 2.0 is slated for deprecation in 2026. Starting February 2025, we will stop accepting new Azure DevOps OAuth apps. Learn more in our blog post.

Microsoft Entra ID OAuth

Building on a new platform can be overwhelming. In this guide to building a Microsoft Entra app for Azure DevOps, we collect helpful links that might be useful to kicking off the OAuth app development process on Microsoft Entra. For folks migrating from Azure DevOps OAuth to Microsoft Entra OAuth, we provide helpful tips to consider during your migration effort.

Azure DevOps OAuth

For existing apps, see the Azure DevOps OAuth app guide. You can also manage which Azure DevOps apps are authorized to access your resources.

Scopes

Developers are expected to specify what scopes they require from their users. The same scopes are available on both OAuth models. The following scopes are available via delegated (on-behalf-of user) flows only. To find out what scopes you need for your app, look under the `scopes` header on the API Reference page for each API you're using.

Some scopes might be inclusive of other scopes, for example, `vso.code_manage` includes `vso.code_write`. For example, many scopes inherit from `vso.profile`. Consider what is the minimal number of scopes you need when requesting scope consent from users.

> **Note**
>
> Scopes only enable access to REST APIs and select Git endpoints. SOAP API access isn't supported.

Expand table

Category	Scope	Name	Description	Inherits From
Advanced Security	`vso.advsec`	AdvancedSecurity (read)	Grants the ability to read alerts, result instances, analysis result instances.	
	`vso.advsec_write`	AdvancedSecurity (read and write)	Grants the ability to upload analyses in sarif	`vso.advsec`
	`vso.advsec_manage`	AdvancedSecurity (read, write, and manage)	Grants the ability to upload analyses in sarif	`vso.advsec_write`
Agent Pools	`vso.agentpools`	Agent Pools (read)	Grants the ability to view tasks, pools, queues, agents, and currently running or	

Category	Scope	Name	Description	Inherits From
			recently completed jobs for agents.	
	`vso.agentpools_manage`	Agent Pools (read, manage)	Grants the ability to manage pools, queues, and agents.	`vso.agentpools`
	`vso.environment_manage`	Environment (read, manage)	Grants the ability to manage pools, queues, agents, and environments.	`vso.agentpools_manage`
Analytics	`vso.analytics`	Analytics (read)	Grants the ability to query analytics data.	
Auditing	`vso.auditlog`	Audit Log (read)	Grants the ability to read the auditing log to users.	
	`vso.auditstreams_manage`	Audit Streams (read)	Grants the ability to manage auditing streams to users.	`vso.auditlog`
Build	`vso.build`	Build (read)	Grants the ability to access build artifacts, including build results, definitions, and requests, and the ability to receive notifications about build events via service hooks.	`vso.hooks_write`
	`vso.build_execute`	Build (read and execute)	Grants the ability to access build artifacts, including build results, definitions, and requests, and the ability to queue a build, update	`vso.build`

Category	Scope	Name	Description	Inherits From
			build properties, and the ability to receive notifications about build events via service hooks.	
Code	`vso.code`	Code (read)	Grants the ability to read source code and metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to search code and get notified about version control events via service hooks.	`vso.hooks_write`
	`vso.code_write`	Code (read and write)	Grants the ability to read, update, and delete source code, access metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to create and manage pull requests and code reviews and to receive notifications about version control events via service hooks.	`vso.code`
	`vso.code_manage`	Code (read, write, and manage)	Grants the ability to read, update, and delete source	`vso.code_write`

Category	Scope	Name	Description	Inherits From
			code, access metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to create and manage code repositories, create and manage pull requests and code reviews, and to receive notifications about version control events via service hooks.	
	`vso.code_full`	Code (full)	Grants full access to source code, metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to create and manage code repositories, create and manage pull requests and code reviews, and to receive notifications about version control events via service hooks. Also includes limited support for Client OM APIs.	`vso.code_manage`
	`vso.code_status`	Code (status)	Grants the ability to read and write	

Category	Scope	Name	Description	Inherits From
			commit and pull request status.	
Connected Server	`vso.connected_server`	Connected Server	Grants the ability to access endpoints needed from an on-premises connected server.	
Entitlements	`vso.entitlements`	Entitlements (Read)	Provides read only access to licensing entitlements endpoint to get account entitlements.	
	`vso.memberentitlementmanagement`	MemberEntitlement Management (read)	Grants the ability to read users, their licenses as well as projects and extensions they can access.	
	`vso.memberentitlementmanagement_write`	MemberEntitlement Management (write)	Grants the ability to manage users, their licenses as well as projects and extensions they can access.	`vso.memberentitlementmanagement`
Extensions	`vso.extension`	Extensions (read)	Grants the ability to read installed extensions.	`vso.profile`
	`vso.extension_manage`	Extensions (read and manage)	Grants the ability to install, uninstall, and perform other administrative actions on installed extensions.	`vso.extension`
	`vso.extension.data`	Extension data (read)	Grants the ability to read data (settings and	`vso.profile`

Category	Scope	Name	Description	Inherits From
			documents) stored by installed extensions.	
	`vso.extension.data_write`	Extension data (read and write)	Grants the ability to read and write data (settings and documents) stored by installed extensions.	`vso.extension.data`
Github Connections	`vso.githubconnections`	GitHub Connections (read)	Grants the ability to read GitHub connections and GitHub repositories data.	
	`vso.githubconnections_manage`	GitHub Connections (read and manage)	Grants the ability to read and manage GitHub connections and GitHub repositories data	`vso.githubconnections`
Graph & identity	`vso.graph`	Graph (read)	Grants the ability to read user, group, scope, and group membership information.	
	`vso.graph_manage`	Graph (manage)	Grants the ability to read user, group, scope and group membership information, and to add users, groups, and manage group memberships.	`vso.graph`
	`vso.identity`	Identity (read)	Grants the ability to read identities and groups.	
	`vso.identity_manage`	Identity (manage)	Grants the ability to read, write, and manage	`vso.identity`

Category	Scope	Name	Description	Inherits From
			identities and groups.	
Machine Group	`vso.machinegroup_manage`	Deployment group (read, manage)	Provides ability to manage deployment group and agent pools.	`vso.agentpools_manage`
Marketplace	`vso.gallery`	Marketplace	Grants read access to public and private items and publishers.	`vso.profile`
	`vso.gallery_acquire`	Marketplace (acquire)	Grants read access and the ability to acquire items.	`vso.gallery`
	`vso.gallery_publish`	Marketplace (publish)	Grants read access and the ability to upload, update, and share items.	`vso.gallery`
	`vso.gallery_manage`	Marketplace (manage)	Grants read access and the ability to publish and manage items and publishers.	`vso.gallery_publish`
Notifications	`vso.notification`	Notifications (read)	Provides read access to subscriptions and event metadata, including filterable field values.	`vso.profile`
	`vso.notification_write`	Notifications (write)	Provides read and write access to subscriptions and read access to event metadata, including filterable field values.	`vso.notification`
	`vso.notification_manage`	Notifications (manage)	Provides read, write, and management	`vso.notification_write`

Category	Scope	Name	Description	Inherits From
			access to subscriptions and read access to event metadata, including filterable field values.	
	`vso.notification_diagnostics`	Notifications (diagnostics)	Provides access to notification-related diagnostic logs and provides the ability to enable diagnostics for individual subscriptions.	`vso.notification`
Packaging	`vso.packaging`	Packaging (read)	Grants the ability to read feeds and packages.	`vso.profile`
	`vso.packaging_write`	Packaging (read and write)	Grants the ability to create and read feeds and packages.	`vso.packaging`
	`vso.packaging_manage`	Packaging (read, write, and manage)	Grants the ability to create, read, update, and delete feeds and packages.	`vso.packaging_write`
Pipeline Resources	`vso.pipelineresources_use`	Pipeline Resources (use)	Grants the ability to approve a pipeline's request to use a protected resource: agent pool, environment, queue, repository, secure files, service connection, and variable group.	
	`vso.pipelineresources_manage`	Pipeline Resources (use and manage)	Grants the ability to manage a	`vso.pipelineresources_manage`

Category	Scope	Name	Description	Inherits From
			protected resource or a pipeline's request to use a protected resource: agent pool, environment, queue, repository, secure files, service connection, and variable group.	
Project and Team	`vso.project`	Project and team (read)	Grants the ability to read projects and teams.	
	`vso.project_write`	Project and team (read and write)	Grants the ability to read and update projects and teams.	`vso.project`
	`vso.project_manage`	Project and team (read, write and manage)	Grants the ability to create, read, update, and delete projects and teams.	`vso.project_write`
Release	`vso.release`	Release (read)	Grants the ability to read release artifacts, including releases, release definitions and release environment.	`vso.profile`
	`vso.release_execute`	Release (read, write and execute)	Grants the ability to read and update release artifacts, including releases, release definitions and release environment, and the ability to queue a new release.	`vso.release`

Category	Scope	Name	Description	Inherits From
	`vso.release_manage`	Release (read, write, execute and manage)	Grants the ability to read, update, and delete release artifacts, including releases, release definitions and release environment, and the ability to queue and approve a new release.	`vso.release_manage`
Secure Files	`vso.securefiles_read`	Secure Files (read)	Grants the ability to read secure files.	
	`vso.securefiles_write`	Secure Files (read, create)	Grants the ability to read and create secure files.	`vso.securefiles_read`
	`vso.securefiles_manage`	Secure Files (read, create, and manage)	Grants the ability to read, create, and manage secure files.	`vso.securefiles_write`
Security	`vso.security_manage`	Security (manage)	Grants the ability to read, write, and manage security permissions.	
Service Connections	`vso.serviceendpoint`	Service Endpoints (read)	Grants the ability to read service endpoints.	`vso.profile`
	`vso.serviceendpoint_query`	Service Endpoints (read and query)	Grants the ability to read and query service endpoints.	`vso.serviceendpoint`
	`vso.serviceendpoint_manage`	Service Endpoints (read, query and manage)	Grants the ability to read, query, and manage service endpoints.	`vso.serviceendpoint_query`
Service Hooks	`vso.hooks`	Service hooks (read)	Grants the ability to read service hook subscriptions and metadata,	`vso.profile`

Category	Scope	Name	Description	Inherits From
			including supported events, consumers, and actions. (No longer public.)	
	`vso.hooks_write`	Service hooks (read and write)	Grants the ability to create and update service hook subscriptions and read metadata, including supported events, consumers, and actions. (No longer public.)	`vso.hooks`
	`vso.hooks_interact`	Service hooks (interact)	Grants the ability to interact and perform actions on events received via service hooks. (No longer public.)	`vso.profile`
Settings	`vso.settings`	Settings (read)	Grants the ability to read settings.	
	`vso.settings_write`	Settings (read and write)	Grants the ability to create and read settings.	
Symbols	`vso.symbols`	Symbols (read)	Grants the ability to read symbols.	`vso.profile`
	`vso.symbols_write`	Symbols (read and write)	Grants the ability to read and write symbols.	`vso.symbols`
	`vso.symbols_manage`	Symbols (read, write and manage)	Grants the ability to read, write, and manage symbols.	`vso.symbols_write`
Task Groups	`vso.taskgroups_read`	Task Groups (read)	Grants the ability to read	

Category	Scope	Name	Description	Inherits From
			task groups.	
	`vso.taskgroups_write`	Task Groups (read, create)	Grants the ability to read and create task groups.	`vso.taskgroups_read`
	`vso.taskgroups_manage`	Task Groups (read, create and manage)	Grants the ability to read, create and manage taskgroups.	`vso.taskgroups_write`
Team Dashboard	`vso.dashboards`	Team dashboards (read)	Grants the ability to read team dashboard information.	
	`vso.dashboards_manage`	Team dashboards (manage)	Grants the ability to manage team dashboard information.	`vso.dashboards`
Test Management	`vso.test`	Test management (read)	Grants the ability to read test plans, cases, results and other test management related artifacts.	`vso.profile`
	`vso.test_write`	Test management (read and write)	Grants the ability to read, create, and update test plans, cases, results and other test management related artifacts.	`vso.test`
Threads	`vso.threads_full`	PR threads	Grants the ability to read and write to pull request comment threads.	
Tokens	`vso.tokens`	Delegated Authorization Tokens	Grants the ability to manage delegated authorization tokens to users.	

Category	Scope	Name	Description	Inherits From
	`vso.tokenadministration`	Token Administration	Grants the ability to manage (view and revoke) existing tokens to organization administrators.	
User Profile	`vso.profile`	User profile (read)	Grants the ability to read your profile, accounts, collections, projects, teams, and other top-level organizational artifacts.	
	`vso.profile_write`	User profile (write)	Grants the ability to write to your profile.	`vso.profile`
Variable Groups	`vso.variablegroups_read`	Variable Groups (read)	Grants the ability to read variable groups.	
	`vso.variablegroups_write`	Variable Groups (read, create)	Grants the ability to read and create variable groups.	`vso.variablegroups_read`
	`vso.variablegroups_manage`	Variable Groups (read, create and manage)	Grants the ability to read, create and manage variable groups.	`vso.variablegroups_write`
Wiki	`vso.wiki`	Wiki (read)	Grants the ability to read wikis, wiki pages and wiki attachments. Also grants the ability to search wiki pages.	
	`vso.wiki_write`	Wiki (read and write)	Grants the ability to read, create and updates wikis, wiki pages	`vso.wiki`

Category	Scope	Name	Description	Inherits From
			and wiki attachments.	
Work Items	`vso.work`	Work items (read)	Grants the ability to read work items, queries, boards, area and iterations paths, and other work item tracking related metadata. Also grants the ability to execute queries, search work items and to receive notifications about work item events via service hooks.	`vso.hooks_write`
	`vso.work_write`	Work items (read and write)	Grants the ability to read, create, and update work items and queries, update board metadata, read area and iterations paths other work item tracking related metadata, execute queries, and to receive notifications about work item events via service hooks.	`vso.work`
	`vso.work_full`	Work items (full)	Grants full access to work items, queries, backlogs, plans, and work item tracking metadata. Also provides the ability to	`vso.work_write`

Category	Scope	Name	Description	Inherits From
			receive notifications about work item events via service hooks.	
User Impersonation	`user_impersonation`	User Impersonation	Have full access to Visual Studio Team Services REST APIs. **Request and/or consent this scope with caution as it is very powerful!**	

FAQ

Frequently asked questions (FAQs)

Q: Can I use OAuth with my mobile phone app?

A: No. Azure DevOps Services only supports the web server flow, so there's no way to implement OAuth, as you can't securely store the app secret.

Q: Can I use OAuth with the SOAP endpoints and REST APIs?

A: No. OAuth is only supported in the REST APIs.

Related articles

- Choosing the right authentication method
- Building for Azure DevOps with Microsoft Entra OAuth apps
- Using Azure DevOps OAuth

Feedback

Was this page helpful? Yes No

Provide product feedback

Build Azure DevOps integrations with Microsoft Entra OAuth apps

Article • 01/08/2025

Azure DevOps Services

This guide provides information and links to more information on building a Microsoft Entra OAuth app for Azure DevOps. Microsoft Entra ID offers robust identity and access management capabilities, which allow you to authenticate users and perform actions on their behalf using OAuth tokens. Use this information to apply Microsoft Entra OAuth tokens for various app flows, including delegated access and service principal-based access.

Use Microsoft Entra OAuth tokens

The Microsoft identity platform offers many ways to authenticate users via the OAuth 2.0 protocol. In these docs, we use OAuth tokens to colloquially refer to on-behalf-of user flows, also known as delegated flows, for apps that request tokens to perform actions for their users. The rest of this guide offers helpful resources for these app developers.

Another common app flow we support is building on-behalf-of app using service principals and managed identities. Microsoft Entra tokens can also be used for ad-hoc requests with the Azure CLI or git operations through the Git Credential Manager.

> **Important**
>
> When creating a new OAuth 2.0 app, start here with Microsoft Entra OAuth apps, as **Azure DevOps OAuth apps** are planned for deprecation in 2026. **Learn more in our blog post**.

Resources for developers

- Register an application with the Microsoft identity platform
- Add permissions for access to Microsoft Graph: Learn how to add delegated permissions from an Azure resource. Instead of Microsoft Graph, select `Azure DevOps` from the list of resources.

- Read about scopes and permissions in the Microsoft identity platform: Understand the `.default` scope. See the scopes available for Azure DevOps in our list of scopes.
- Request permissions through consent
- Use authentication libraries and code samples
- Manage personal access tokens via API: Use the PAT lifecycle management APIs with Microsoft Entra tokens. Our docs and the associated sample app provide examples for setting up a Microsoft Entra app to use Azure DevOps REST APIs.
- Explore support and help options for developers

Resources for admins

- Understand application management in Microsoft Entra ID
- Add an enterprise application
- Explore the consent experience for applications in Microsoft Entra ID

Tips for building & migrating

> **Note**
>
> Microsoft Entra OAuth apps don't natively support Microsoft account (MSA) users for Azure DevOps REST APIs. If you're building an app that must cater to MSA users or support both Microsoft Entra and MSA users, Azure DevOps OAuth apps remain your best option. We're currently working on native support for MSA users through Microsoft Entra OAuth.

- **Important Azure DevOps IDs:**
  - Microsoft Entra resource identifier: `499b84ac-1321-427f-aa17-267ca6975798`
  - Resource URI: `https://app.vssps.visualstudio.com`
  - Use the `.default` scope when requesting a token with all scopes that the app is permissioned for.
- When you migrate an existing app, you might use Azure DevOps user identifiers that don't exist in Microsoft Entra. Use the ReadIdentities API to resolve and match the different identities used by each identity provider.

Related articles

- Authenticate to Azure DevOps with Microsoft Entra
- Use service principals & managed identities in Azure DevOps

Feedback

Was this page helpful? Yes No

Provide product feedback

Use Azure DevOps OAuth 2.0 to create a web app

Article • 10/29/2024

Azure DevOps Services

> **Important**
>
> Azure DevOps OAuth is slated for deprecation in 2026. This information is for existing Azure DevOps OAuth apps only. To create new apps, use Microsoft Entra ID OAuth to integrate with Azure DevOps. Starting March 2025, we will stop accepting new Azure DevOps OAuth apps. Learn more in our blog post.

Azure DevOps is an identity provider for OAuth 2.0 apps. Our implementation of OAuth 2.0 lets developers authorize their app for users and get access tokens for Azure DevOps resources.

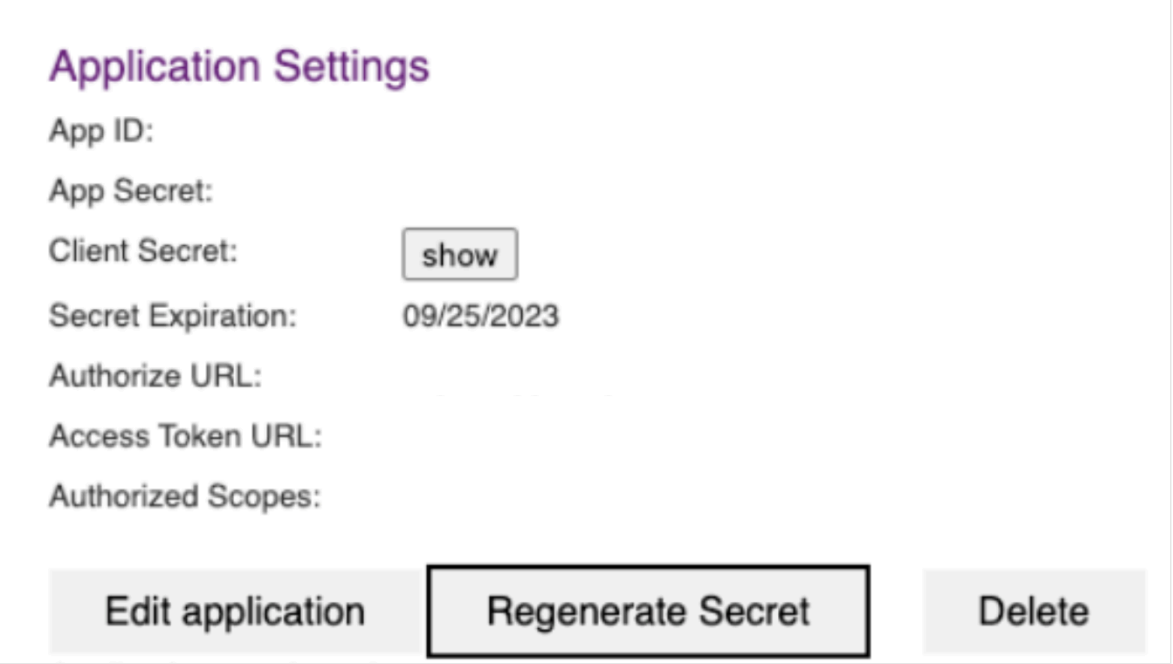
Get started with Azure DevOps OAuth

1. Register your app

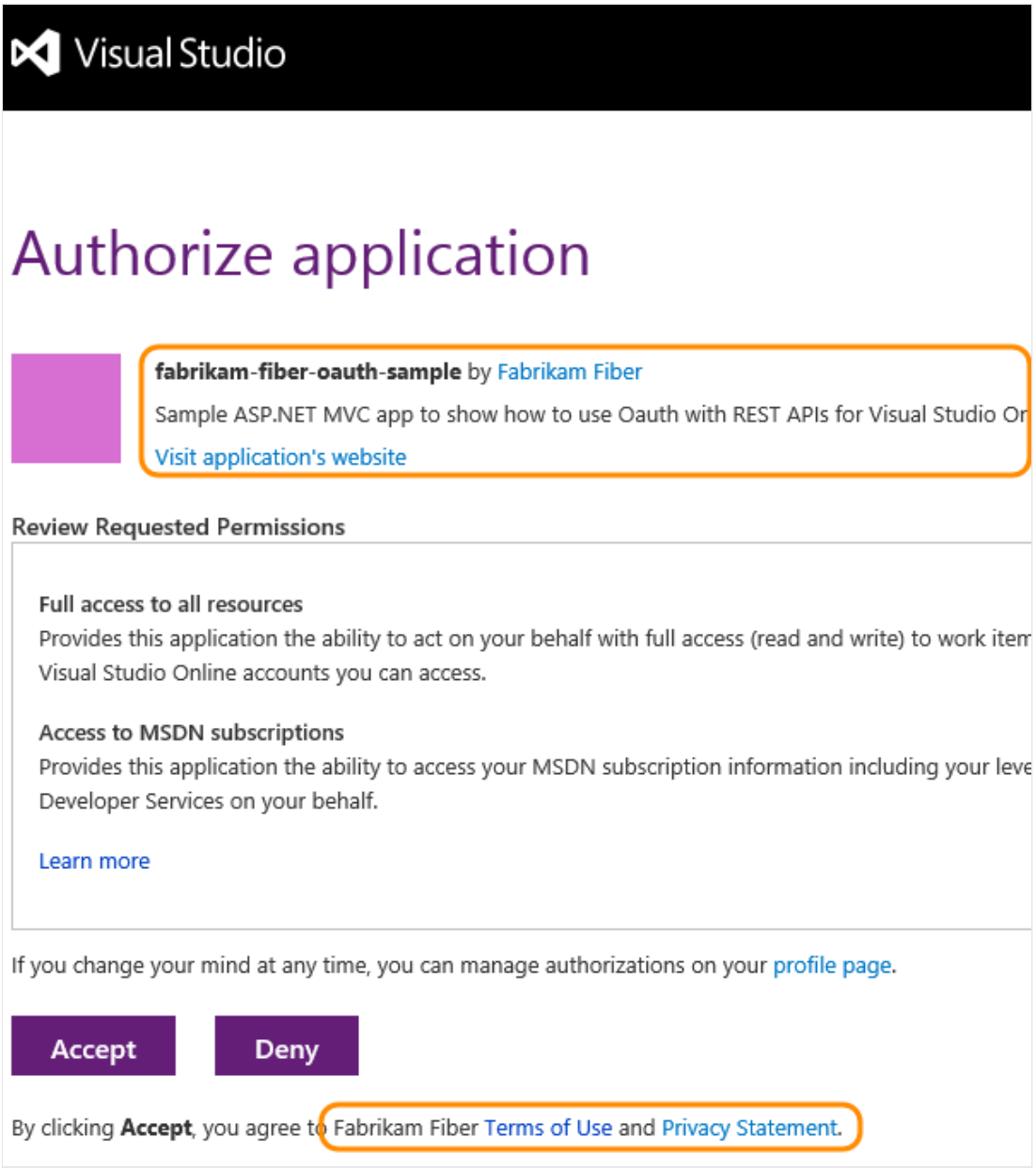
> **Important**
>
> New app creation will be blocked starting February 2025.

1. Go to `https://app.vsaex.visualstudio.com/app/register` to register your app.
2. Select the scopes that your application needs, and then use the same scopes when you authorize your app. If you registered your app using the preview APIs, re-register because the scopes that you used are now deprecated.
3. Select **Create application**.

   The application settings page displays.



- When Azure DevOps Services presents the authorization approval page to your user, it uses your company name, app name, and descriptions. It also uses the URLs for your company web site, app website, and terms of service and privacy statements.



- When Azure DevOps Services asks for a user's authorization, and the user grants it, the user's browser gets redirected to your authorization callback URL with the authorization code. The callback URL must be a secure connection (https) to transfer the code back to the app and exactly match the URL registered in your app. If it doesn't, a 400 error page is displayed instead of a page asking the user to grant authorization to your app.

4. Call the authorization URL and pass your app ID and authorized scopes when you want to have a user authorize your app to access their organization. Call the access token URL when you want to get an access token to call an Azure DevOps Services REST API.

The settings for each app that you register are available from your profile `https://app.vssps.visualstudio.com/profile/view`.

2. Authorize your app

1. If your user didn't authorize your app to access their organization, call the authorization URL. It calls you back with an authorization code, if the user approves the authorization.

```
no-highlight

https://app.vssps.visualstudio.com/oauth2/authorize
        ?client_id={app ID}
        &response_type={Assertion}
        &state={state}
        &scope={scope}
        &redirect_uri={callback URL}
```

Expand table

Parameter	Type	Notes
client_id	GUID	The ID assigned to your app when it was registered.
response_type	string	`Assertion`
state	string	Can be any value. Typically a generated string value that correlates the callback with its associated authorization request.
scope	string	Scopes registered with the app. Space separated. See available scopes.
redirect_uri	URL	Callback URL for your app. **Must exactly match the URL registered with the app**.

2. Add a link or button to your site that takes the user to the Azure DevOps Services authorization endpoint:

```
no-highlight

https://app.vssps.visualstudio.com/oauth2/authorize
        ?client_id=00001111-aaaa-2222-bbbb-3333cccc4444
        &response_type=Assertion
        &state=User1
        &scope=vso.work%20vso.code_write
        &redirect_uri=https://fabrikam.azurewebsites.net/myapp/oauth-
callback
```

Azure DevOps Services asks the user to authorize your app.

Assuming the user accepts, Azure DevOps Services redirects the user's browser to your callback URL, including a short-lived authorization code and the state value provided in the authorization URL:

```
no-highlight
```

```
https://fabrikam.azurewebsites.net/myapp/oauth-callback
        ?code={authorization code}
        &state=User1
```

3. Get an access and refresh token for the user

Use the authorization code to request an access token (and refresh token) for the user. Your service must make a service-to-service HTTP request to Azure DevOps Services.

URL - authorize app

```
no-highlight
```

```
POST https://app.vssps.visualstudio.com/oauth2/token
```

HTTP request headers - authorize app

Expand table

Header	Value
Content-Type	`application/x-www-form-urlencoded`

```
no-highlight
```

```
Content-Type: application/x-www-form-urlencoded
```

HTTP request body - authorize app

```
no-highlight
```

```
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-
bearer&client_assertion={0}&grant_type=urn:ietf:params:oauth:grant-type:jwt-
bearer&assertion={1}&redirect_uri={2}
```

Replace the placeholder values in the previous sample request body:

- **{0}**: URL encoded client secret acquired when the app was registered

- {1}: URL encoded "code" provided via the `code` query parameter to your callback URL
- {2}: callback URL registered with the app

C# example to form the request body - authorize app

no-highlight

```
public string GenerateRequestPostData(string appSecret, string authCode,
string callbackUrl)
{
   return String.Format("client_assertion_type=urn:ietf:params:oauth:client-
assertion-type:jwt-bearer&client_assertion=
{0}&grant_type=urn:ietf:params:oauth:grant-type:jwt-bearer&assertion=
{1}&redirect_uri={2}",
               HttpUtility.UrlEncode(appSecret),
               HttpUtility.UrlEncode(authCode),
               callbackUrl
        );
}
```

Response - authorize app

JSON

```
{
    "access_token": { access token for the user },
    "token_type": { type of token },
    "expires_in": { time in seconds that the token remains valid },
    "refresh_token": { refresh token to use to acquire a new access token }
}
```

> ⓘ **Note**
>
> Securely persist the **refresh_token** so your app doesn't need to prompt the user to authorize again. Access tokens expire quickly and shouldn't be persisted.

4. Use the access token

To use an access token, include it as a bearer token in the Authorization header of your HTTP request:

```
Authorization: Bearer {access_token}
```

For example, the HTTP request to get recent builds for a project:

```
no-highlight

GET https://dev.azure.com/myaccount/myproject/_apis/build-release/builds?
api-version=3.0
Authorization: Bearer {access_token}
```

5. Refresh an expired access token

If a user's access token expires, you can use the refresh token that they acquired in the authorization flow to get a new access token. It's like the original process for exchanging the authorization code for an access and refresh token.

URL - refresh token

```
no-highlight

POST https://app.vssps.visualstudio.com/oauth2/token
```

HTTP request headers - refresh token

Expand table

Header	Value
Content-type	`application/x-www-form-urlencoded`
Content-length	Calculated string length of the request body (see the following example)

```
no-highlight

Content-Type: application/x-www-form-urlencoded
Content-Length: 1654
```

HTTP request body - refresh token

```
no-highlight
```

```
client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-
bearer&client_assertion={0}&grant_type=refresh_token&assertion=
{1}&redirect_uri={2}
```

Replace the placeholder values in the previous sample request body:

- **{0}**: URL encoded client secret acquired when the app was registered
- **{1}**: URL encoded refresh token for the user
- **{2}**: callback URL registered with the app

Response - refresh token

JSON

```
{
    "access_token": { access token for this user },
    "token_type": { type of token },
    "expires_in": { time in seconds that the token remains valid },
    "refresh_token": { new refresh token to use when the token has timed out
}
}
```

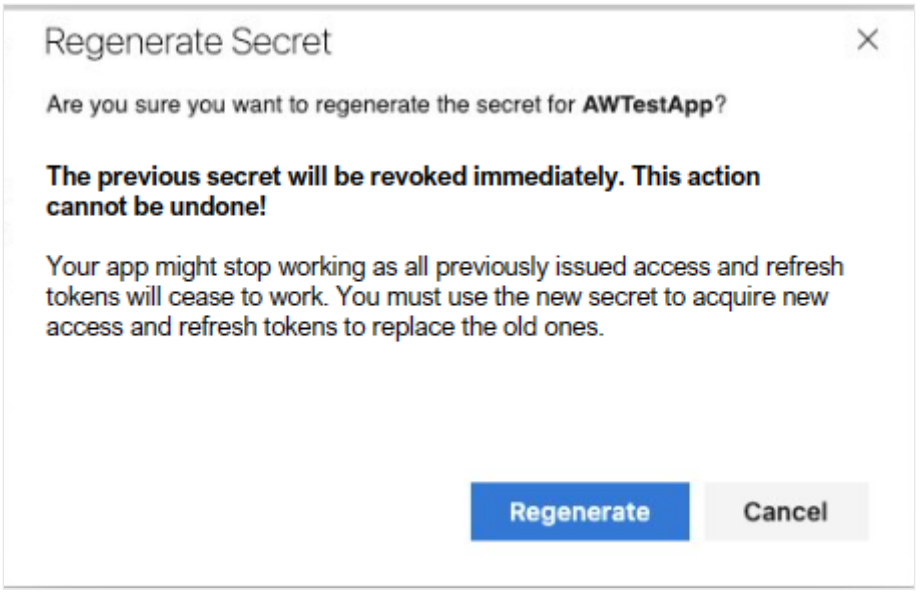
> **Note**
>
> A new refresh token gets issued for the user. Persist this new token and use it the next time you need to acquire a new access token for the user.

Samples

You can find a C# sample that implements OAuth to call Azure DevOps Services REST APIs in our C# OAuth GitHub Sample.

Regenerate client secret

Every five years, your application secret expires. Regenerate your app secret to continue to create and use access tokens and refresh tokens. To do so, select "Regenerate secret," which then confirms that you want to complete this action.

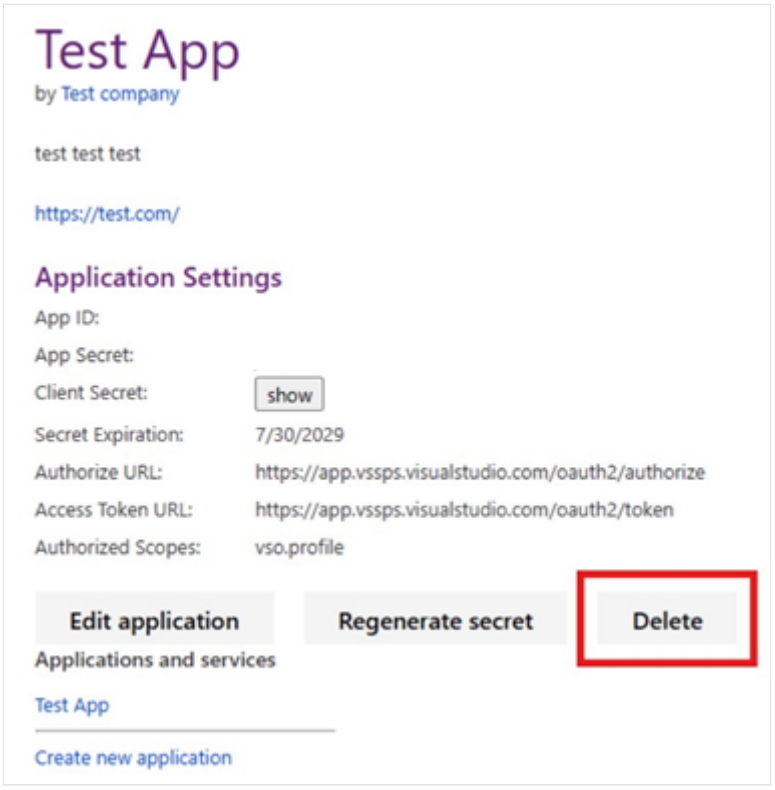


When you confirm that you want to regenerate, the previous app secret no longer works and all previous tokens minted with this secret also stop working. Make sure to time this client secret rotation well to minimize any customer downtime.

Delete your app

If you no longer need your app, delete it from your profile.

1. Go to your profile at: `https://app.vssps.visualstudio.com/profile/view`.
2. Ensure you're on the correct tenant's page by selecting from the dropdown menu under your name in the sidebar.
3. Find the app under the **Applications and services** header on the left sidebar.
4. select "Delete" on the application registration page. A modal appears to confirm your deletion.



5. Once you delete the app registration, the app breaks, and we stop minting new tokens or accepting minted tokens for this app.

Frequently asked questions (FAQs)

Q: Can I use OAuth with my mobile phone app?

A: No. Azure DevOps Services only supports the web server flow, so there's no way to implement OAuth, as you can't securely store the app secret.

Q: What errors or special conditions do I need to handle in my code?

A: Make sure that you handle the following conditions:

- If your user denies your app access, no authorization code gets returned. Don't use the authorization code without checking for denial.
- If your user revokes your app's authorization, the access token is no longer valid. When your app uses the token to access data, a 401 error returns. Request authorization again.

Q: I want to debug my web app locally. Can I use localhost for the callback URL when I register my app?

A: Yes. Azure DevOps Services now allows localhost in your callback URL. Ensure you use `https://localhost` as the beginning of your callback URL when you register your app.

Q: I get an HTTP 400 error when I try to get an access token. What might be wrong?

A: Check that you set the content type to application/x-www-form-urlencoded in your request header.

Q: I get an HTTP 401 error when I use an OAuth-based access token, but a PAT with the same scope works fine. Why?

A: Verify that your organization's admin didn't disable **Third-party application access via OAuth** at `https://dev.azure.com/{your-org-name}/_settings/organizationPolicy`. In this scenario, the flow to authorize an app and generate an access token works, but all REST APIs return only an error, such as `TF400813: The user "<GUID>" is not authorized to access this resource.`

Related articles

- Choosing the right authentication method
- Use Microsoft Entra ID OAuth
- Default permissions and access for Azure DevOps
- Manage authorizations

Feedback

Was this page helpful? Yes No

Provide product feedback

Authenticate and secure web extensions

Article • 07/11/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

This article pertains only to authentication and security for *web extensions*, and not Pipelines task extensions or service endpoint extensions. For those tasks, you can use the Publish to Azure Service Bus Task.

Call REST APIs from your extension

Most extensions need to call Azure DevOps REST APIs on behalf of the current user.

- If you're using the provided `JavaScript REST clients`, authentication is automatically handled for you. These clients request an access token from the core SDK and set it in the Authorization header of the request.
- If you're not using the provided clients, you need to request a token from the `Core SDK` and set it in the Authorization header of your request:

  JavaScript

  ```
  import * as SDK from "azure-devops-extension-sdk";
  import { getAccessToken } from "azure-devops-extension-sdk";

  SDK.init();

  getAccessToken().then((token) => {
      // Format the auth header
      const authHeader = `Bearer ${token}`;

      // Add token as an Authorization header to your request
      console.log(authHeader);
  });
  ```

> **Tip**
>
> Check out our newest documentation on extension development using the Azure DevOps Extension SDK.

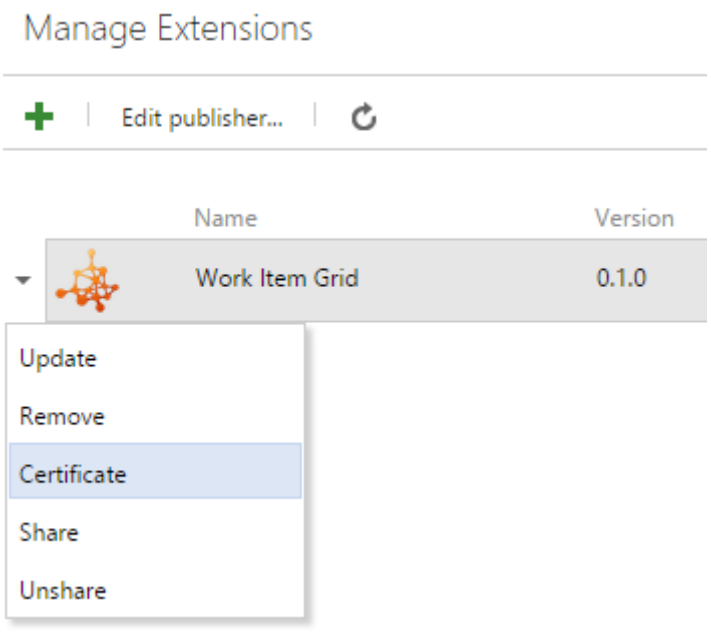
Authenticate requests to your service

A common scenario is making calls to a back-end service from an extension. To verify these calls are coming from your extension running in Azure DevOps and to authenticate the current user, and other context information, a special type of token is provided to your extension. This token contains information about the caller and a signature that you can validate to ensure the request originated from your extension.

Get your extension's key

Your extension's unique key, generated when the extension is published, can be used to verify the authenticity of requests made from your extension.

To obtain this key, go to the extension management portal, right-click on a published extension, and then select **Certificate**.



> ⚠ **Warning**
>
> Scope changes in an extension cause the certificate to change. If you make changes to the scope, you need a new extension key.

Generate a token to provide to your service

1. The Core SDK `getAppToken` method returns a promise that, when resolved, contains a token signed with your extension's certificate.

   JavaScript

```
import * as SDK from "azure-devops-extension-sdk";
import { getAppToken } from "azure-devops-extension-sdk";

SDK.init();

getAppToken().then((token) => {
// Add token to your request
console.log(token);
});
```

2. Pass this token to your service as a query parameter or request header.

Parse and validate the token

Here is a sample of parsing the token. First, download and store the secret for your extension from your publisher page. This secret must be available to your application.

.NET Framework

Do the following task to add one reference to get the sample to compile.

Open the NuGet Package Manager and add a reference to `System.IdentityModel.Tokens.Jwt`. This sample was built with version 6.8.0 of this package.

C#

```
using System;
using System.IdentityModel.Tokens.Jwt;
using Microsoft.IdentityModel.Tokens;

namespace TokenSample
{
    class Program
    {
        static void Main(string[] args)
        {
            string secret = ""; // Load your extension's secret
            string issuedToken = ""; // Token you are validating

            var validationParameters = new TokenValidationParameters()
            {
                IssuerSigningKey = new
SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(secret)),
                ValidateIssuer = false,
                RequireSignedTokens = true,
                RequireExpirationTime = true,
                ValidateLifetime = true,
```

```
                ValidateAudience = false,
                ValidateActor = false
            };

            SecurityToken token = null;
            var tokenHandler = new JwtSecurityTokenHandler();
            var principal = tokenHandler.ValidateToken(issuedToken,
validationParameters, out token);

            // Use the principal object as needed
            Console.WriteLine(principal.Identity.Name);
        }
    }
}
```

.NET Core - WebAPI

Do the following task to add one reference to get this sample to compile.

Open the NuGet Package Manager and add a reference to `System.IdentityModel.Tokens.Jwt`. This sample was built with version 5.1.4 of this package.

Startup.cs

C#

```
using System.Text;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.IdentityModel.Tokens;

namespace TokenSample.Core.API
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();

            string _secret = "ey9asfasdmax..<the secret key downloaded from
```

```
the Azure DevOps Services publisher page>.9faf7eh";


services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
                    .AddJwtBearer((o) =>
                    {
                        o.TokenValidationParameters = new
TokenValidationParameters()
                        {
                            IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_secret)),
                            ValidateIssuer = false,
                            ValidateAudience = false,
                            ValidateActor = false,
                            RequireSignedTokens = true,
                            RequireExpirationTime = true,
                            ValidateLifetime = true
                        };
                    });
        }

        public void Configure(IApplicationBuilder app, IWebHostEnvironment
env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

            app.UseAuthentication();
            app.UseAuthorization();
            app.UseRouting();
            app.UseStaticFiles();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}
```

Your API Controllers:

C#

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

[Route("api/[controller]")]
[Authorize]
public class SampleLogicController : ControllerBase
{
```

```
    // ...
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback

Use the Cross-platform CLI for Azure DevOps using personal access tokens

Article • 01/28/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Use personal access tokens (PATs) with the cross-platform CLI for Azure DevOps.

> **ⓘ Important**
>
> We don't recommend using basic authentication or PATs. Instead, we recommend using Microsoft Entra-based authentication, if you're interacting with Microsoft Entra accounts.

Prerequisites

- Latest version of Node.js source code
- **Cross-platform CLI for Azure DevOps**
  - Install **tfx-cli** using `npm`, a component of Node.js by running:

```
no-highlight

  npm i -g tfx-cli
```

  For more information about **tfx-cli**, see the Node CLI for Azure DevOps on GitHub.

Personal access token

Create a personal access token and paste it into the sign-in command.

```
~$ tfx login
Copyright Microsoft Corporation

> Service URL: {url}
> Personal access token: xxxxxxxxxxxx
Logged in successfully
```

See the following examples of valid URLs:

- `https://marketplace.visualstudio.com`
- `https://youraccount.visualstudio.com/DefaultCollection`

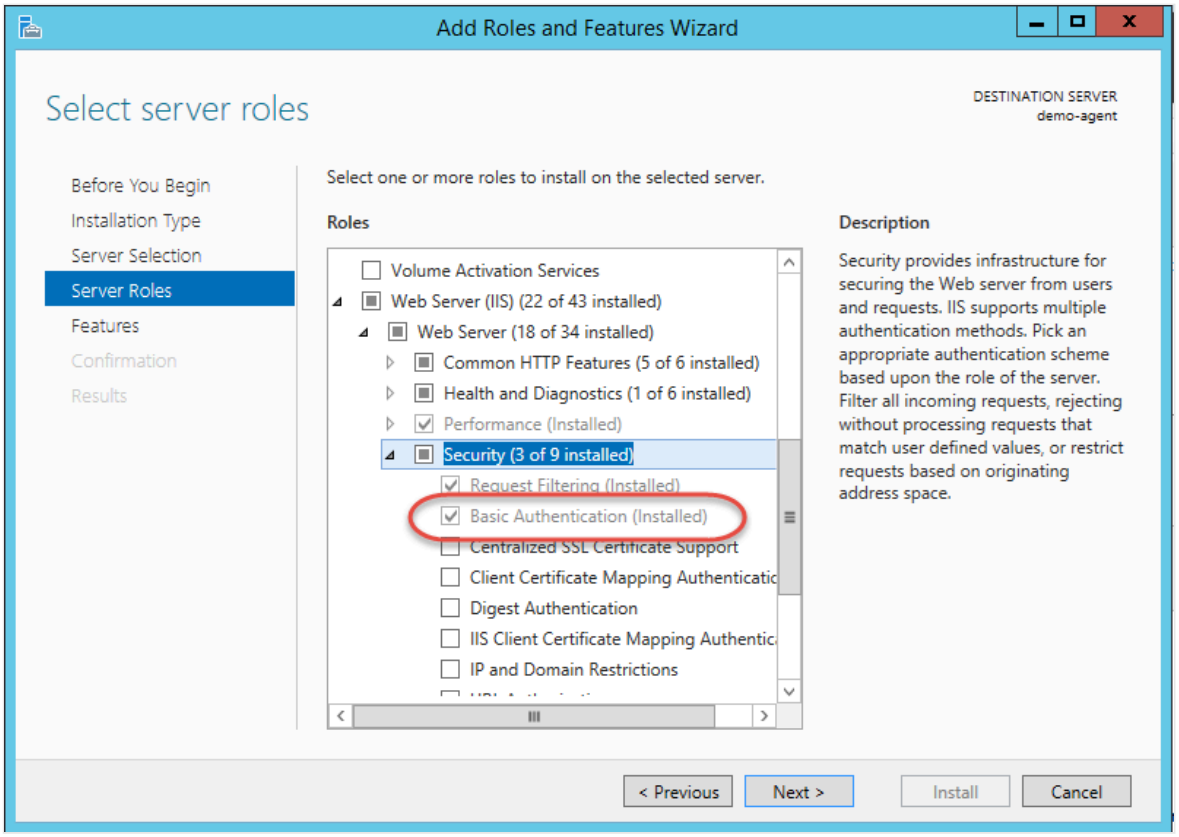
Configure for Basic authentication

> **⚠ Warning**
>
> We don't recommend basic authentication. Turning on IIS basic authentication causes various issues, and you should use personal access tokens (PATs) instead. For example, if you turn on IIS basic authentication, GIT command line stops working.

Follow these steps to enable basic auth for your Azure DevOps instance:

> **⚠ Warning**
>
> Basic authentication sends usernames and passwords in plaintext. Consider configuring Azure DevOps Server to use SSL to enable secure communication when using basic auth.

1. Install the `Basic Authentication` feature for IIS in Server Manager.



2. Open IIS Manager and expand to the `Azure DevOps Server` website, double-click the `Authentication` tile in the Features view.

3. Choose `Basic Authentication` in the list of authentication methods. Choose `Enable` in the right hand column. You should now see `Basic Authentication` enabled.

> ⓘ **Note**
>
> Leave the **domain** and **realm** settings for Basic Authentication empty.

tfx sign in with Basic authentication

Now you can start to use `tfx` against your server. Sign in before you issue commands.

1. Enter the following command:

```
no-highlight
```

```
tfx login --auth-type basic
```

2. Add your service url.

3. Add your username. Use `domain\user` (for example, fabrikam\peter). If you're working on a workgroup machine, use `machinename\user`.

4. Add your password. Enter the password for the username that you previously entered.

You can now use any other tfx command.

no-highlight

```
> tfx login --auth-type basic
Copyright Microsoft Corporation

Enter service url > http://localhost:8080/tfs/defaultcollection
Enter username > fabfiber\peter
Enter password > *******
logged in successfully
```

Feedback

Was this page helpful? Yes No

Provide product feedback

Develop a web extension

Article • 01/18/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Use extensions to enhance Azure DevOps with new web experiences, dashboard widgets, build tasks, and more. You can develop extensions using standard technologies like HTML, JavaScript, and CSS. This tutorial guides you through creating a web extension for Azure DevOps.

> **Tip**
>
> Explore the extension samples and the newest documentation on extension development using the Azure DevOps Extension SDK.

Prerequisites

You must have the following permission and installations.

- You must be an organization owner. If you don't have an organization, you can create an organization for free.
- Install Node.js.
- Install the extension packaging tool (TFX). Run `npm install -g tfx-cli` from a command prompt, which you use to package your extension later on.

Create a directory and manifest

An extension is composed of a set of files that includes a required manifest file. You package it into a .vsix file and publish to the Visual Studio Marketplace.

1. Create a directory to hold the files needed for your extension:

```
mkdir my-first-extension
```

2. From this directory, initialize a new npm package manifest:

```
npm init -y
```

This file describes the libraries required by your extension.

3. Install the Microsoft VSS Web Extension SDK package and save it to your npm package manifest:

```
npm install azure-devops-extension-sdk --save
```

This SDK includes a JavaScript library that provides APIs required for communicating with the page your extension is embedded in.

4. Create an extension manifest file named `vss-extension.json` at the root of your extension directory with the following content:

JSON

```
{
    "manifestVersion": 1,
    "id": "my-first-extension",
    "publisher": "",
    "version": "1.0.0",
    "name": "My First Extension",
    "description": "A sample Visual Studio Services extension",
    "public": false,
    "categories": ["Azure Repos"],
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "contributions": [
        {
            "id": "my-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-code-web.code-hub-group"
            ],
            "properties": {
                "name": "My Hub",
                "uri": "my-hub.html"
            }
        }
    ],
    "files": [
        {
            "path": "my-hub.html",
```

```
            "addressable": true
        },
        {
            "path": "node_modules/azure-devops-extension-sdk",
            "addressable": true,
            "packagePath": "lib"
        }
    ]
}
```

> **Note**
>
> The `public` property controls whether the extension is visible to everyone on the Visual Studio Marketplace. Keep your extensions private during development.

5. Create a file named `my-hub.html` at the root of your extension directory with the following content, which is for the view (also known as a hub) contributed into the web experience.

HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/require.js/2.3.6/require.mi
n.js"></script>
    <script>
        window.requirejs.config({
            enforceDefine: true,
            paths: {
                'SDK': './lib/SDK.min'
            }
        });
        window.requirejs(['SDK'], function (SDK) {
            if (typeof SDK !== 'undefined') {
                console.log("SDK is defined. Trying to initialize...");
                SDK.init();
                SDK.ready().then(() => {
                    console.log("SDK is ready");
                    document.getElementById("name").innerText =
SDK.getUser().displayName;
                });
            } else {
                console.log('SDK is not defined');
            }
        });
    </script>
```

```
    <style>
        body {
            background-color: rgb(0, 67, 117);
            color: white;
            margin: 10px;
            font-family: "Segoe UI VSS (Regular)","-apple-
system",BlinkMacSystemFont,"Segoe UI",sans-serif;
        }
    </style>
</head>
<body>
    <h1>Hello, <span id="name"></span></h1>
</body>
</html>
```

6. Your extension directory should look like the following example.

```
|-- my-hub.html
|-- node_modules
    |-- @types
    |-- azure-devops-extension-sdk
|-- package.json
|-- vss-extension.json
```

Need help? Post questions to the Azure DevOps Services Developer Community.

Next steps

Package and publish your extension

Related articles

- Developer Formula Design System
- Contribution model

Feedback

Was this page helpful? Yes No

Provide product feedback

Extensibility points

Article • 09/13/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Extensions enhance the Azure DevOps user experience by contributing new capabilities. In this article, we highlight the most common extensibility points that your extension can target. For more information about the Azure DevOps extensibility model, see the Contribution model.

The Contributions Guide extension is a sample extension. Install this extension into your organization. Once it's installed, you see the extensibility points that are available. We recommend you install this extension into a personal or test organization. The source for this extension is also available. Also, find a sample of a repository creation extension point here.

For more information, see the following references:

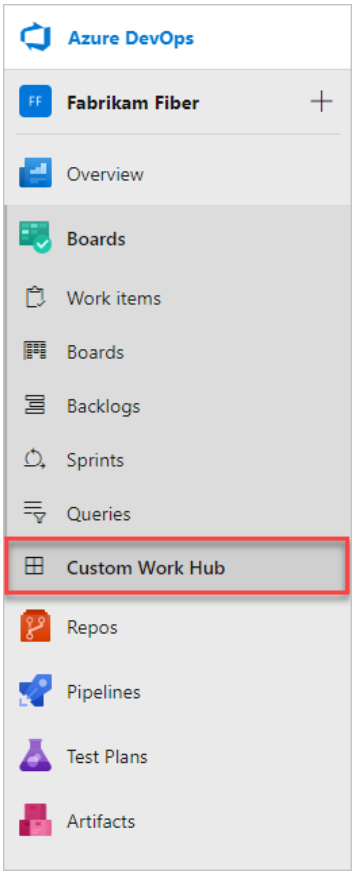
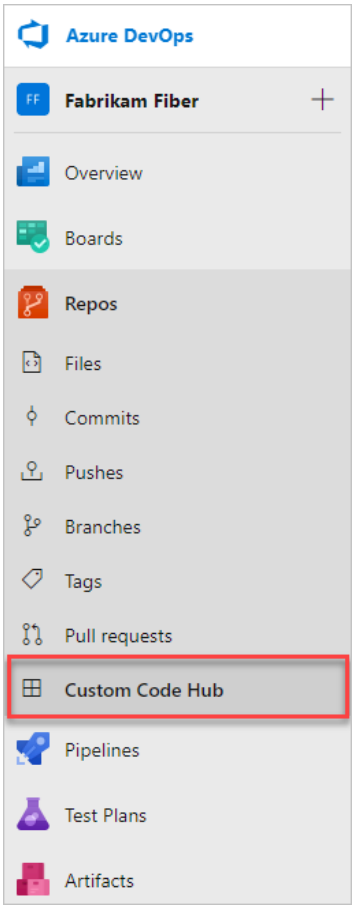
- azure-devops-extension-api
- azure-devops-extension-sdk
- azure-devops-extension-sample
- installed extension api

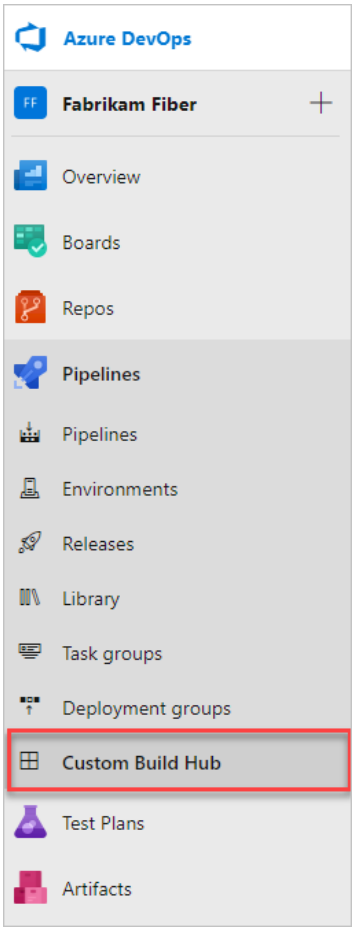
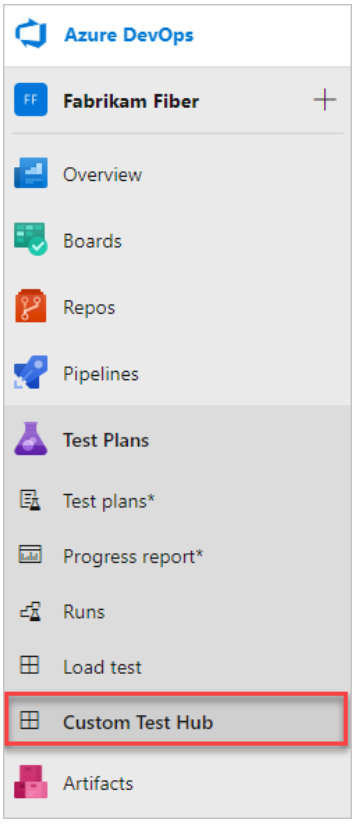
Hubs and hub groups

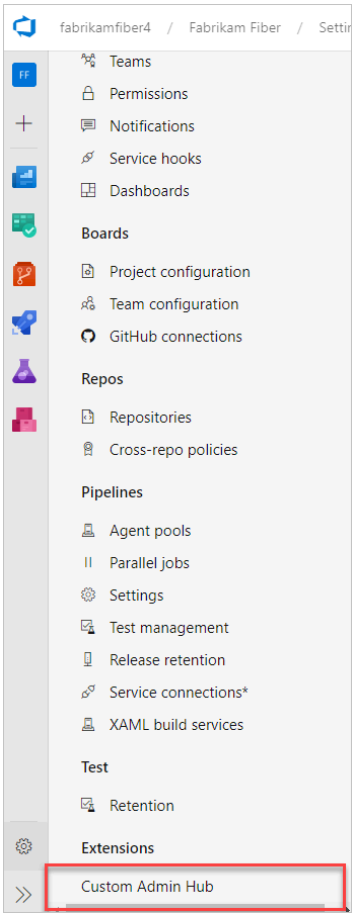
Hubs and hub groups are the primary navigation elements in Azure DevOps. **Files**, **Releases**, **Backlogs**, and **Queries** are examples of hubs. A hub belongs to a hub group. The **Files** hub, for example, belongs to the project-level **Azure Repos** hub group. Hub groups can exist at the organization or collection level or the project level. Most extensions contribute to the project level.

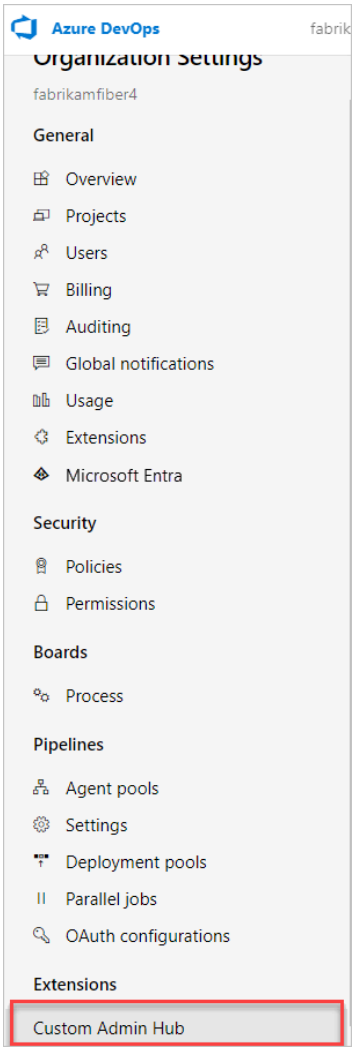
The following table describes the most common hub groups in Azure DevOps that you can contribute hubs to.

Expand table

Name	ID	Level	Preview image
Azure Boards	`ms.vss-work-web.work-hub-group`	Project/team	
Azure Repos	`ms.vss-code-web.code-hub-group`	Project/team	

Name	ID	Level	Preview image
Azure Pipelines	`ms.vss-build-web.build-release-hub-group`	Project/team	
Azure Test Plans	`ms.vss-test-web.test-hub-group`	Project/team	

Name	ID	Level	Preview image
Project settings	`ms.vss-web.project-admin-hub-group`	Project	

Name	ID	Level	Preview image
Organization settings	`ms.vss-web.collection-admin-hub-group`	Organization or collection	

Example

The following example shows how to contribute a hub to the Code hub group:

JSON

```
{
    ...
    "contributions": [
        {
            "id": "my-custom-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-code-web.code-hub-group"
            ],
            "properties": {
                "name": "Code Hub",
                "order": 30,
                "uri": "/views/code/custom.html"
            }
        }
```

```
    ]
}
```

- `ms.vss-web.hub` is the type of this contribution. This type is defined in the `vss-web` extension published under the `ms` publisher. This type declares optional and required properties that are required by contributions of this type (for example, name, order, and so on).
- `ms.vss-code-web.code-hub-group` is the full ID of the hub group contribution this hub is targeting. This contribution is declared in the `vss-code-web` extension published under the `ms` publisher
- `my-custom-hub` is the short ID of this contribution; `{publisherId}.{extensionId}.my-custom-hub` is the full ID

Add an icon to your menu or toolbar

Add an icon property, so it can be used directly by name.

We recommend providing your own icon. Using your own icon example:

"properties":

```
            "name": "Sample hub",
            "uri": "dist/Hub/Hub.html",
            "icon": "asset://static/sample-icon.png",
            "supportsMobile": true
        }
```

Using the Office UI Fabric Icons example:

"properties":

```
             "iconName": "Code",
             "name": "Code Hub",
             "order": 30,
             "uri": "/views/code/custom.html"
         }
```

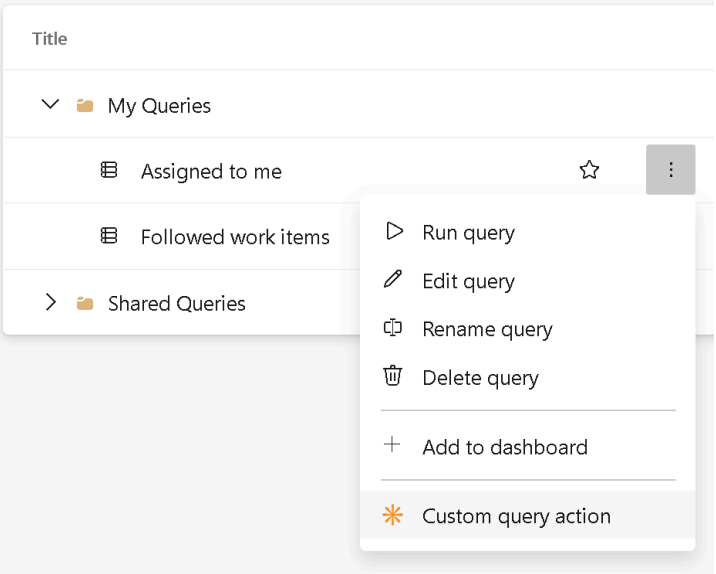
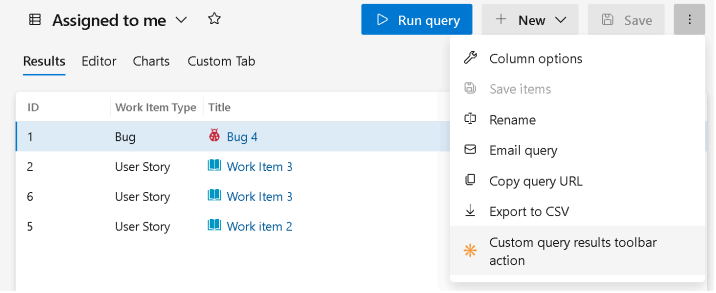
Settings for menus and toolbars

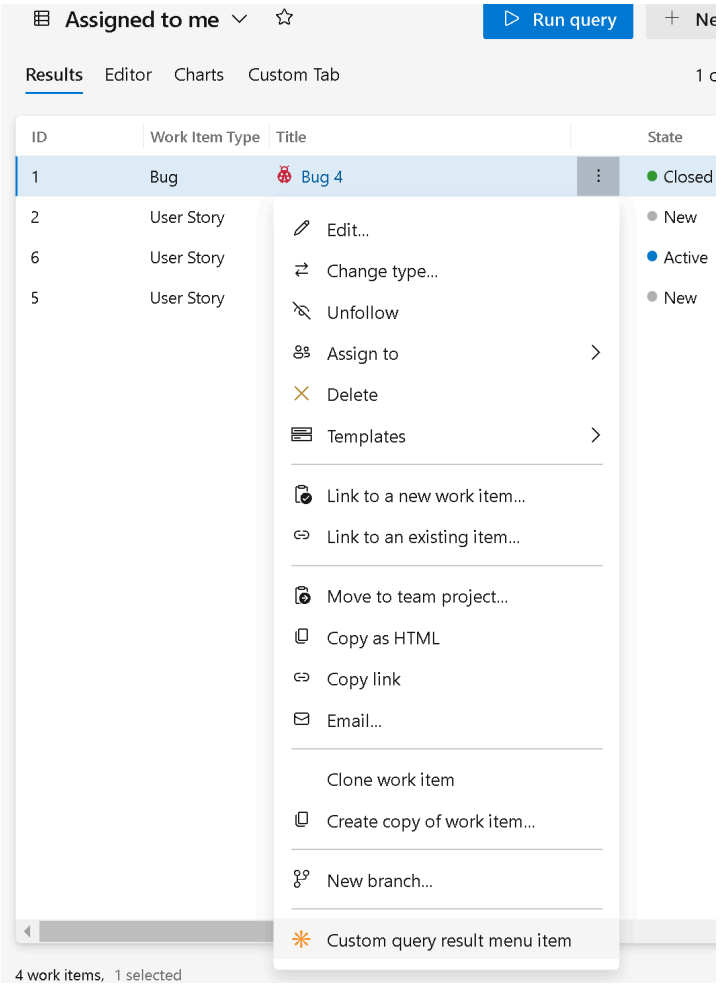
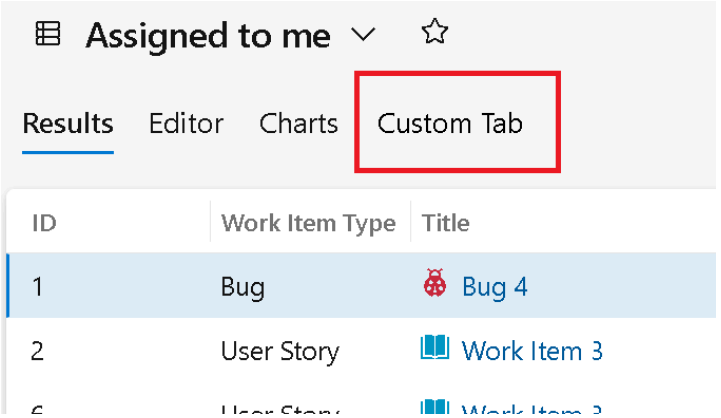
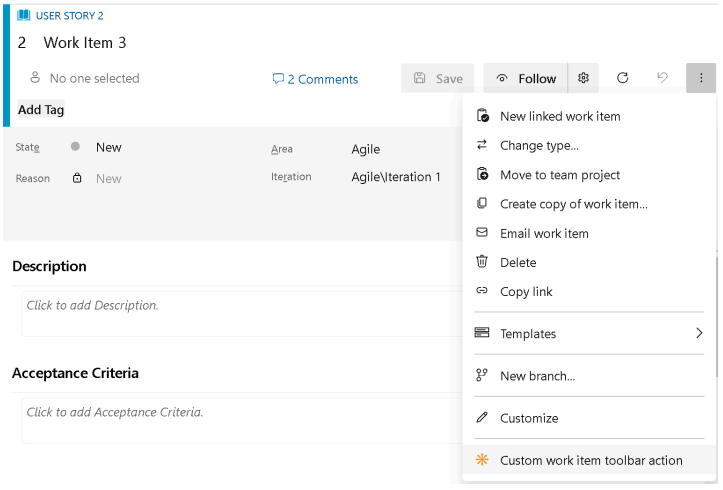
Expand table

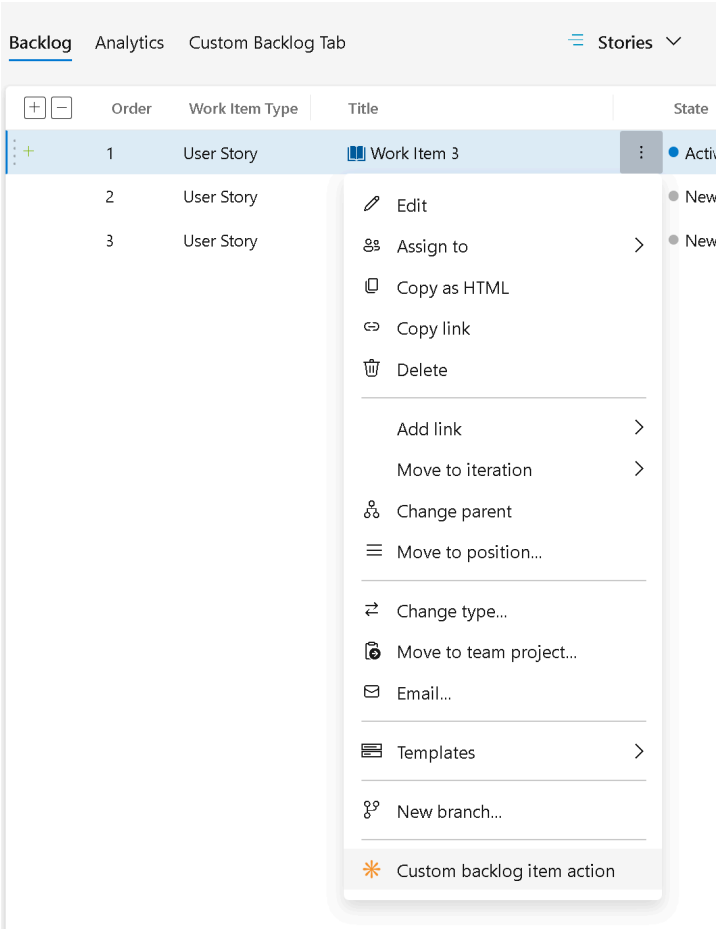
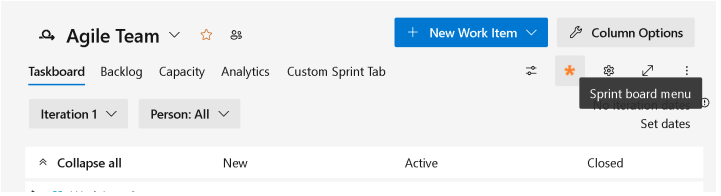
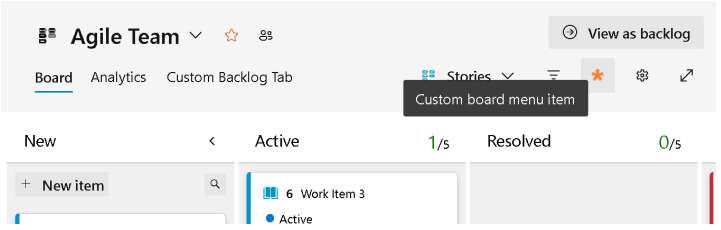
Name	Target ID
Organization/collection overview toolbar	`ms.vss-admin-web.collection-overview-toolbar-menu`
Collection overview projects grid	`ms.vss-admin-web.projects-grid-menu`
Project overview toolbar	`ms.vss-admin-web.project-overview-toolbar-menu`
Project overview teams grid	`ms.vss-admin-web.teams-grid-menu`

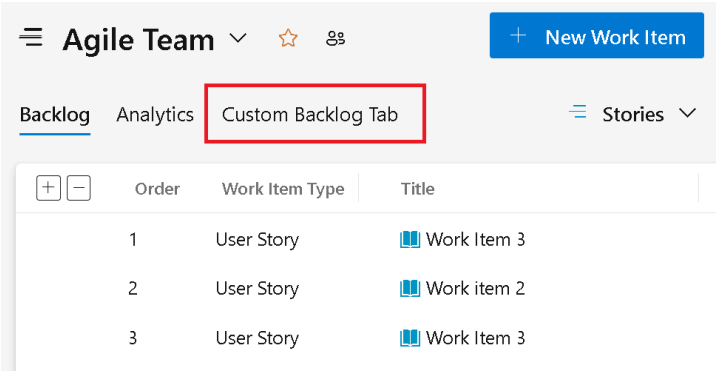
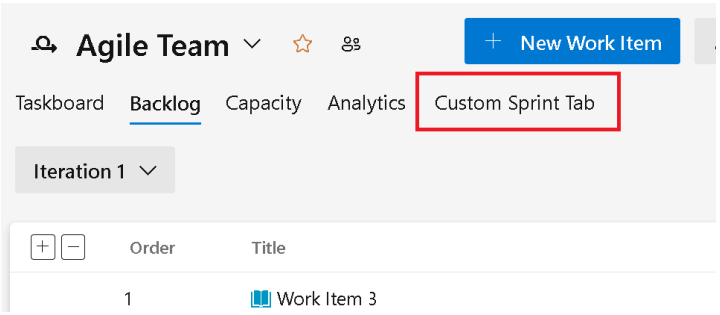
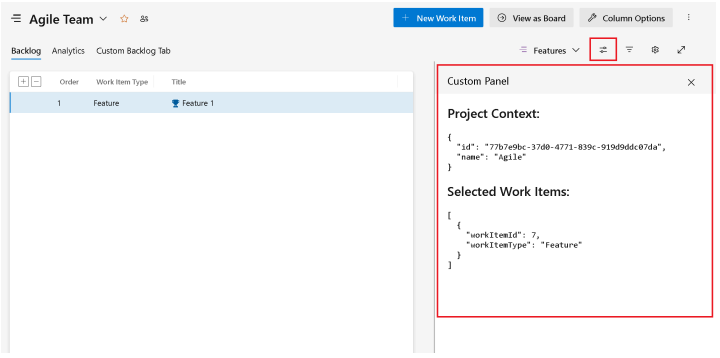
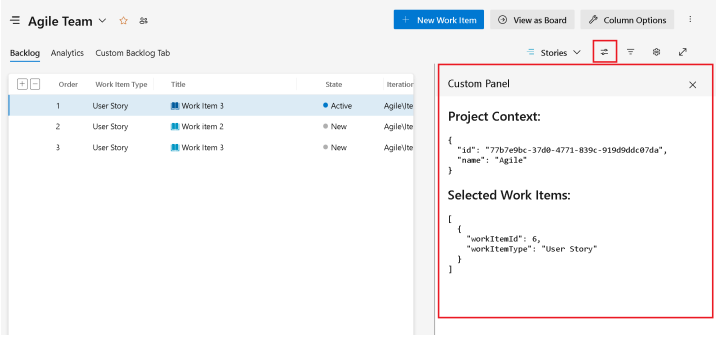
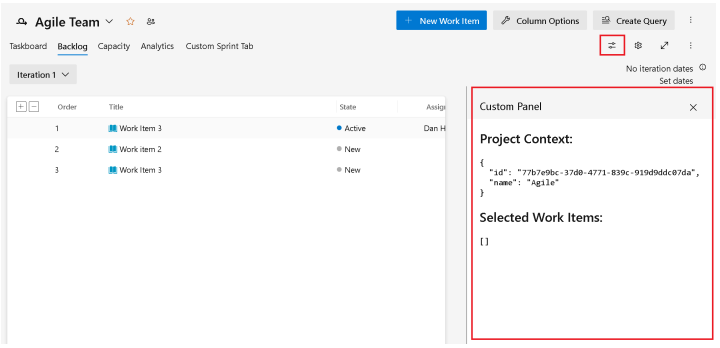
Azure Boards menu and toolbar

Expand table

Name	Target ID	Preview image
Work item query menu	`ms.vss-work-web.work-item-query-menu`	
Work item query results toolbar menu	`ms.vss-work-web.work-item-query-results-toolbar-menu`	

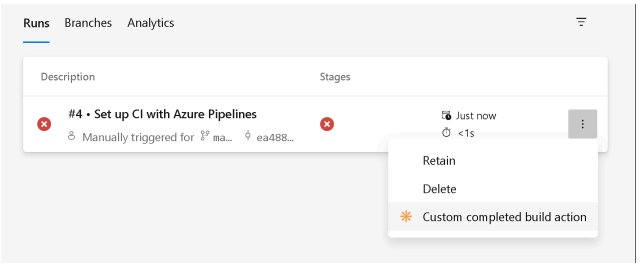
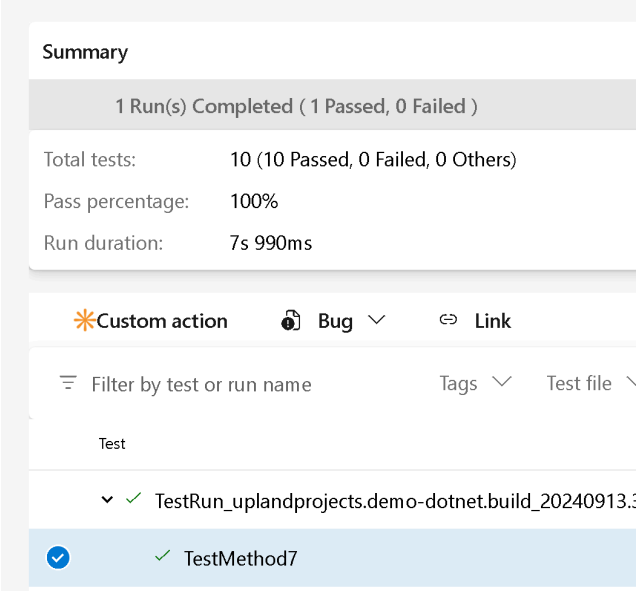
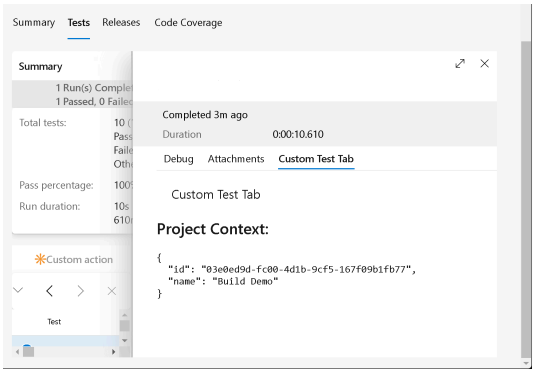
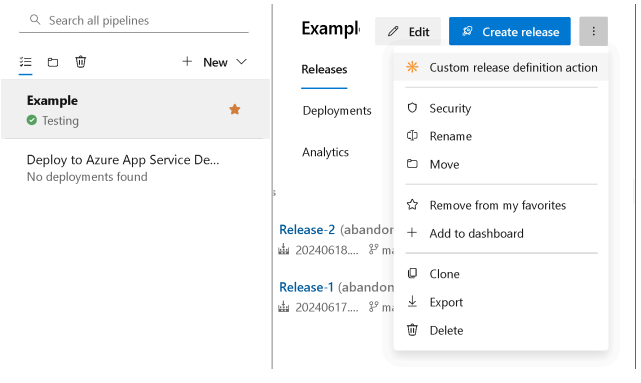
Name	Target ID	Preview image
Work item query results menu item	`ms.vss-work-web.query-result-work-item-menu`	
Work item query results tab	`ms.vss-work-web.query-tabs`	
Work item for context menu	`ms.vss-work-web.work-item-toolbar-menu`	

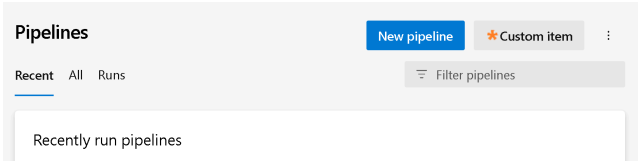
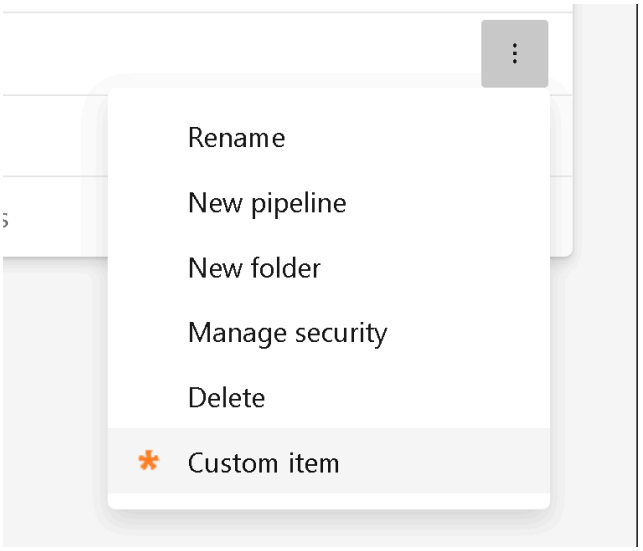
Name	Target ID	Preview image
Backlog item menu	`ms.vss-work-web.backlog-item-menu`	
Sprint board pivot filter menu	`ms.vss-work-web.sprint-board-pivot-filter-menu`	
Board pivot filter menu	`ms.vss-work-web.backlog-board-pivot-filter-menu`	
Card menu	`ms.vss-work-web.backlog-board-card-item-menu`	

Name	Target ID	Preview image
Product backlog tab	`ms.vss-work-web.product-backlog-tabs`	
Iteration backlog tab	`ms.vss-work-web.iteration-backlog-tabs`	
Portfolio backlog pane	`ms.vss-work-web.portfolio-backlog-toolpane`	
Product backlog pane	`ms.vss-work-web.requirement-backlog-toolpane`	
Iteration backlog pane	`ms.vss-work-web.iteration-backlog-toolpane`	

Azure Pipelines menu and toolbar

Expand table

Name	Target ID	Preview
Completed build menu	`ms.vss-build-web.completed-build-menu`	
Test results toolbar action	`ms.vss-test-web.test-results-actions-menu`	
Test result details tab	`ms.vss-test-web.test-result-details-tab-items`	
Release pipeline explorer context menu	`ms.vss-releaseManagement-web.release-definition-explorer-context-menu`	

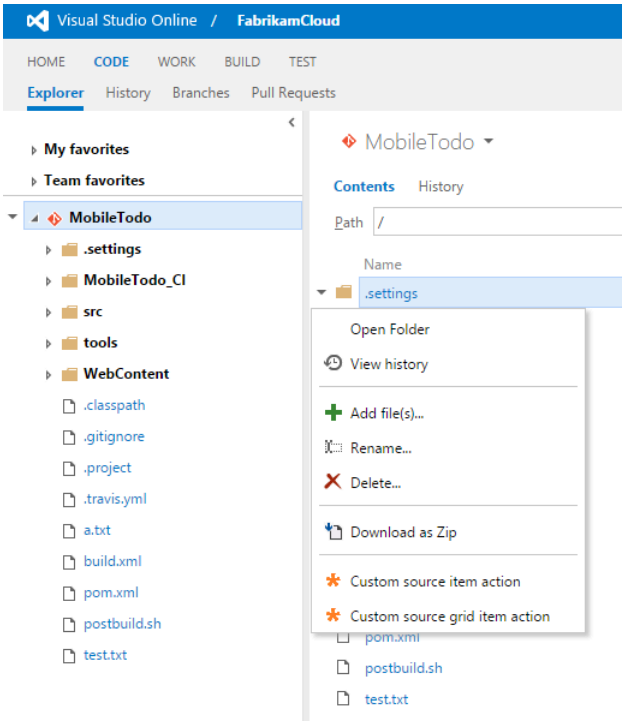
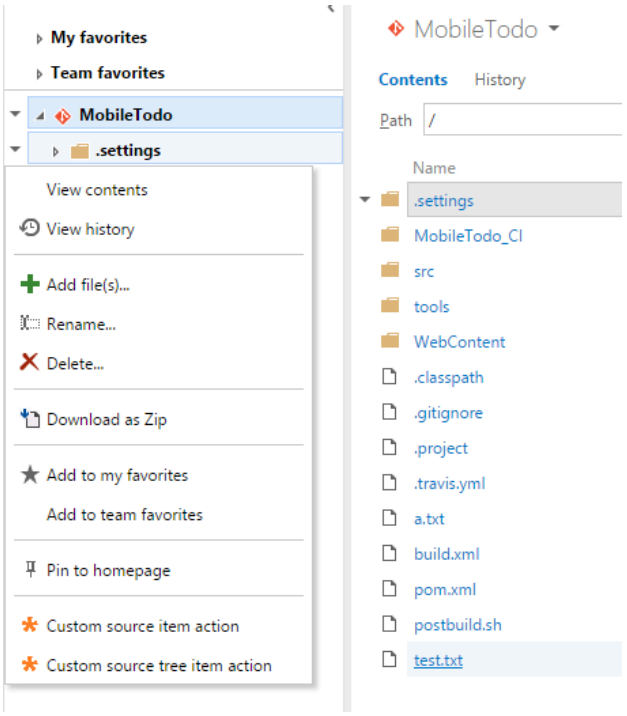
Name	Target ID	Preview
Pipeline details view, header button	`ms.vss-build-web.pipelines-header-menu`	
Pipeline details view, folder context menu	`ms.vss-build-web.pipelines-folder-menu`	

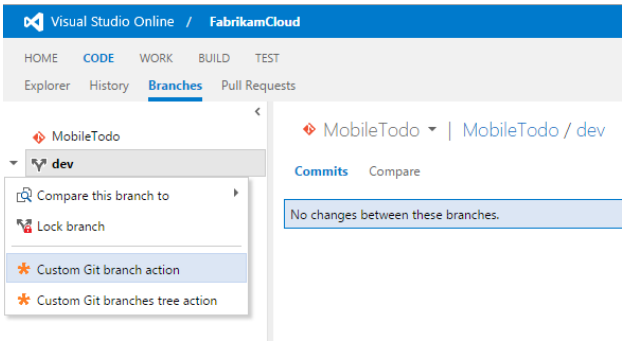
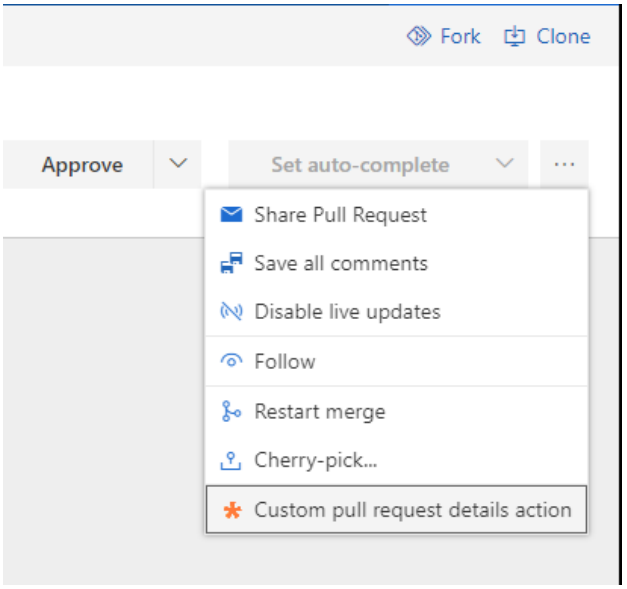
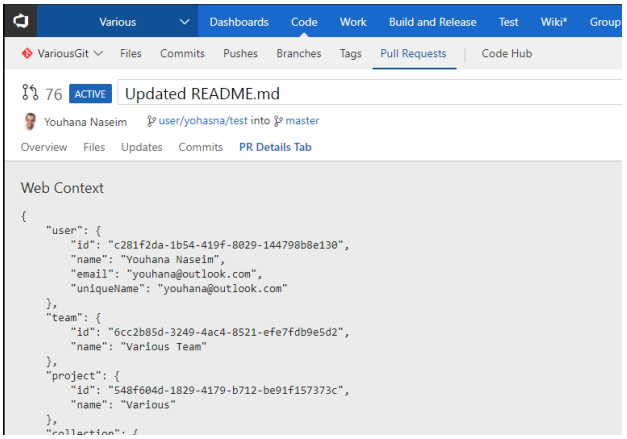
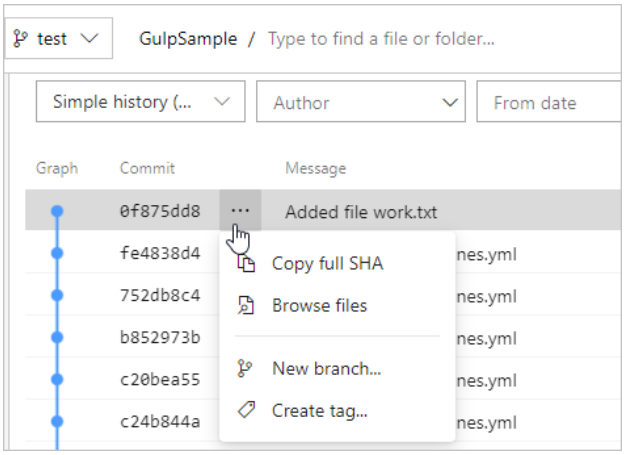
Azure Pipelines tasks

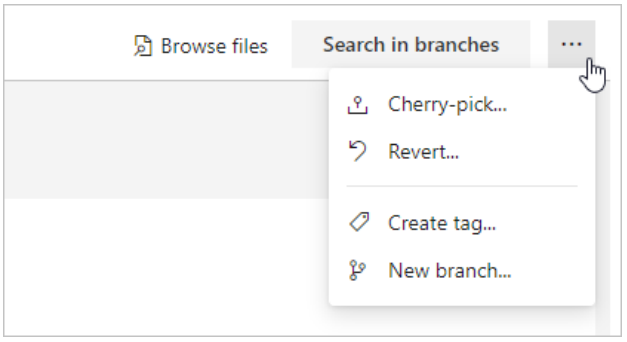
Tasks perform work in a build or release. For more information, learn how to contribute a build or release task.

Azure Repos menu and toolbar

Expand table

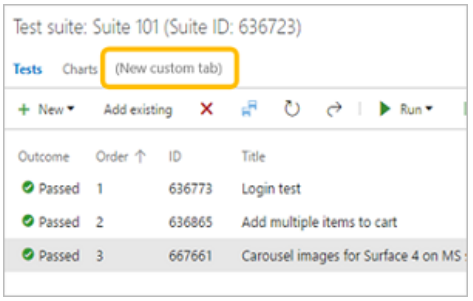
Name	Target ID	Preview image
Source item (grid) menu	`ms.vss-code-web.source-grid-item-menu`	
Source item (tree) menu	`ms.vss-code-web.source-tree-item-menu`	
Source item (grid and tree) menu	`ms.vss-code-web.source-item-menu`	
Change list item menu	`ms.vss-code-web.change-list-item-menu`	
Change list summary item menu	`ms.vss-code-web.change-list-summary-item-menu`	

Name	Target ID	Preview image
Git branches tree menu	`ms.vss-code-web.git-branches-tree-menu`	
Git pull request actions menu	`ms.vss-code-web.pull-request-action-menu`	
Git pull request tabs (pivots)	`ms.vss-code-web.pr-tabs`	
Git commit listing menu	`ms.vss-code-web.git-commit-list-menu`	

Name	Target ID	Preview image
Git commit detail menu	`ms.vss-code-web.git-commit-details-menu`	

Azure Test Plans menu and toolbar

Expand table

Name	Target ID	Preview image
Test run grid menu	`ms.vss-test-web.test-run-grid-menu`	
Test plan suites tree menu	`ms.vss-test-web.test-plans-suites-context`	
Test plan hub pivot tab	`ms.vss-test-web.test-plan-pivot-tabs`	

Other extensibility points

- **Dashboard widget**: An extension can contribute a new type of widget that can be added by users to a dashboard. Learn how to contribute a dashboard widget.
- **Work item form**: The work item form is enhanced by extensions with new sections, tabs, actions, and custom field renderers. For more information, learn how to extend the work item form.
- **Service hooks**: A "consumer" is the service that events are sent to in Service Hooks. An extension can contribute consumer services. These services get configured by a user (or programmatically), to send events to that service. For more information, see contribute a service hooks consumer.
- **Features**:

Name: Preview feature (hosted only)
Target ID: ms.vss-web.managed-features

Feedback

Was this page helpful? Yes No

Provide product feedback

Contribution model

Article • 10/17/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Extensions add new capabilities into the system by providing contributions and by defining ways for other extensions to plug in to them. A contribution type defines something that can be added to the system. A contribution is a specific instance of a contribution type. For example, `hub` is a contribution type defined by a core Microsoft-provided extension. The `Explorer` hub under the Build hub group is a specific contribution of type `hub`.

For more information, see the following references:

- Extensibility points
- azure-devops-extension-api
- azure-devops-extension-sdk

Contribution types

A **contribution type** defines a contract that all contributions to the system of that type must adhere to. A contribution type can also extend from another contribution type. The following examples of contribution types define the properties set by instances of that type:

- `hub`
- `action`
- `build-task`

Each property definition specifies the following aspects:

- Property type, for example string, boolean, and so on.
- Whether the property is required
- Default value, if not specified by a contribution (optional).

Example

Here's an example of a contribution type declaration in an extension manifest:

```
JavaScript
```

```
{
    ...
    "contributionTypes": [
        {
            "id": "hub",
            "name": "Web Access Hub",
            "description": "A hub that appears in the hubs menu at the top
of web pages.",
            "properties": {
                "name": {
                    "description": "The text to display for the hub",
                    "type": "string",
                    "required": true
                },
                "uri": {
                    "description": "URI of the contents of the hub page",
                    "type": "string",
                    "required": true
                },
                "order": {
                    "description": "An optional ordering value which can
indicate in which position to place the hub within the hub group",
                    "type": "integer"
                }
                ...
            }
        }
    ]
}
```

Contributions

A **contribution** is an instance of a contribution type. For example, the `Queries` hub under the Work hub group is a contribution of type `hub` and the `Publish Test Results` build task is a contribution of type `build-task`.

All contributions must specify a type and specify values for any properties required by that contribution type.

Example

Here's an example of a hub contribution declaration in an extension manifest:

JavaScript

```
{
    "contributions": [
        {
```

```
            "id": "build-explorer-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                ".build-hub-group"
            ],
            "properties": {
                "name": "Explorer",
                "uri": "/_build",
                "order": 22
            }
        }
    ]
}
```

Target contributions

A contribution can **target** one or more other contributions, which creates a relationship between the contribution and each of its targets. The system can discover contributions for the target at runtime. For example, a `hub` contribution (`Explorer`) might target a specific `hub-group` contribution (`Build`).

JSON

```
{
    "id": "build-explorer-hub",
    "type": "ms.vss-web.hub",
    "targets": [
        ".build-hub-group"
    ]
}
```

When the hub group renders, the system can query for all hub contributions that target the hub group to know which hubs to render.

Identify contributions and types

Every contribution and contribution type must have a unique ID within the extension it's declared in.

A *full* contribution identifier includes the following items, which you separate with a dot `.`:

- Publisher ID
- Extension ID
- Contribution/type ID

For example: `ms.vss-web.hub` is the full identifier for the following contribution:

- Publisher ID: `ms`
- Extension ID: `vss-web`
- Contribution/type ID: `hub`

You can use *relative* contribution references within an extension manifest for a contribution's reference to another contribution or contribution type within that same extension. In this case, the publisher and extension IDs aren't included, and the ID is a dot `.` followed by the contribution ID. For example, `.hub` may be used within the `vss-web` extension mentioned previously as a shortcut for `ms.vss-web.hub`.

Add a custom pipelines task extension

Article • 10/10/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Learn how to install extensions to your organization for custom build or release tasks in Azure DevOps. For more information, see What is Azure Pipelines?

> **Note**
>
> This article covers agent tasks in agent-based extensions. For more information on server tasks and server-based extensions, see the Server Task GitHub Documentation.

Prerequisites

To create extensions for Azure DevOps, you need the following software and tools.

Expand table

Software/tool	Information
Azure DevOps organization	Create an organization.
A text editor	For many procedures, we use Visual Studio Code, which provides intellisense and debugging support. Download the latest version.
Node.js	Download the latest version.
npmjs.com 4.0.2 or newer	TypeScript Compiler. Download the latest version.
tfx-cli	Package your extension with Cross-platform CLI for Azure DevOps. using `npm`, a component of Node.js, by running `npm i -g tfx-cli`.
Azure DevOps extension SDK	Install the azure-devops-extension-sdk package.
A `home` directory for your project	The `home` directory of a build or release task extension should look like the following example after you complete the steps in this article.

```
|--- README.md
|--- images
    |--- extension-icon.png
|--- buildandreleasetask            // where your task scripts are placed
|--- vss-extension.json             // extension's manifest
```

> **Important**
>
> The dev machine must run the latest version of Node to ensure that the written code is compatible with the production environment on the agent and the latest non-preview version of `azure-pipelines-task-lib`. Update your task.json file as per the following command:

```
"execution": {
   "Node20_1": {
     "target": "index.js"
   }
 }
```

1. Create a custom task

Do every part of this procedure within the `buildandreleasetask` folder.

> **Note**
>
> This example walk-through uses Windows with PowerShell. We made it generic for all platforms, but the syntax for getting environment variables is different. If you're using a Mac or Linux, replace any instances of `$env:<var>=<val>` with `export <var>=<val>`.

Create task scaffolding

1. Create the folder structure for the task and install the required libraries and dependencies.

2. Open a PowerShell command window, go to your `buildandreleasetask` folder, and run the following command.

```
npm init --yes
```

`npm init` creates the `package.json` file. We added the `--yes` parameter to accept all of the default `npm init` options.

> **Tip**
>
> The agent doesn't automatically install the required modules because it's expecting your task folder to include the node modules. To mitigate this, copy the `node_modules` to `buildandreleasetask`. As your task gets bigger, it's easy to exceed the size limit (50MB) of a VSIX file. Before you copy the node folder, you may want to run `npm install --production` or `npm prune --production`, or you can write a script to build and pack everything.

3. Add `azure-pipelines-task-lib` to your library.

```
npm install azure-pipelines-task-lib --save
```

4. Ensure that TypeScript typings are installed for external dependencies.

```
npm install @types/node --save-dev
npm install @types/q --save-dev
```

5. Create a `.gitignore` file and add node_modules to it. Your build process should do an `npm install` and a `typings install` so that node_modules are built each time and don't need to be checked in.

```
echo node_modules > .gitignore
```

6. Install Mocha as a development dependency.

```
npm install mocha --save-dev -g
npm install sync-request --save-dev
```

```
npm install @types/mocha --save-dev
```

7. Choose TypeScript version 2.3.4 or 4.6.3.

```

npm install typescript@4.6.3 -g --save-dev
```

> **Note**
>
> Make sure that TypeScript is installed globally with npm in your development environment, so the `tsc` command is available. If you skip this step, TypeScript version 2.3.4 gets used by default, and you still have to install the package globally to have the `tsc` command available.

8. Create `tsconfig.json` compiler options. This file ensures that your TypeScript files are compiled to JavaScript files.

```

tsc --init --target es2022
```

Create task

Now that the scaffolding is complete, we can create our custom task.

1. Create a `task.json` file in the `buildandreleasetask` folder. The `task.json` file describes the build/release task and is what the build/release system uses to render configuration options to the user and to know which scripts to execute at build/release time.

2. Copy the following code and replace the `{{placeholders}}` with your task's information. The most important placeholder is the `taskguid`, and it must be unique.

JSON

```
{
 "$schema": "https://raw.githubusercontent.com/Microsoft/azure-pipelines-task-lib/master/tasks.schema.json",
 "id": "{{taskguid}}",
 "name": "{{taskname}}",
```

```
"friendlyName": "{{taskfriendlyname}}",
"description": "{{taskdescription}}",
"helpMarkDown": "",
"category": "Utility",
"author": "{{taskauthor}}",
"version": {
    "Major": 0,
    "Minor": 1,
    "Patch": 0
},
"instanceNameFormat": "Echo $(samplestring)",
"inputs": [
    {
        "name": "samplestring",
        "type": "string",
        "label": "Sample String",
        "defaultValue": "",
        "required": true,
        "helpMarkDown": "A sample string"
    }
],
"execution": {
    "Node20_1": {
        "target": "index.js"
    }
}
}
```

3. Create an `index.ts` file by using the following code as a reference. This code runs when the task gets called.

TypeScript

```
import tl = require('azure-pipelines-task-lib/task');

 async function run() {
    try {
        const inputString: string | undefined =
tl.getInput('samplestring', true);
        if (inputString == 'bad') {
            tl.setResult(tl.TaskResult.Failed, 'Bad input was given');
            return;
        }
        console.log('Hello', inputString);
    }
    catch (err:any) {
        tl.setResult(tl.TaskResult.Failed, err.message);
    }
 }

 run();
```

4. To compile an `index.js` file from `index.ts`, enter "tsc" from the `buildandreleasetask` folder.

task.json components

See the following descriptions of some of the components of the `task.json` file.

Expand table

Property	Description
`id`	A unique GUID for your task.
`name`	Name with no spaces.
`friendlyName`	Descriptive name (spaces allowed).
`description`	Detailed description of what your task does.
`author`	Short string describing the entity developing the build or release task, for example: "Microsoft Corporation."
`instanceNameFormat`	How the task displays within the build/release step list. You can use variable values by using **$(variablename)**.
`groups`	Describes the logical grouping of task properties in the UI.
`inputs`	Inputs to be used when your build or release task runs. This task expects an input with the name **samplestring**.
`execution`	There are multiple execution options for this task, including scripts, like `Node`, `PowerShell`, `PowerShell3`, or `Process`.
`restrictions`	Restrictions being applied to the task about GitHub Codespaces commands task can call, and variables task can set. We recommend that you specify restriction mode for new tasks.

> **Note**
>
> Create an `id` with the following command in PowerShell:
>
> PowerShell
>
> ```
> (New-Guid).Guid
> ```
>
> For more information, see the Build/release task reference.

Run the task

Run the task with `node index.js` from PowerShell.

In the following example, the task fails because inputs weren't supplied (`samplestring` is a required input).

```
node index.js
##vso[task.debug]agent.workFolder=undefined
##vso[task.debug]loading inputs and endpoints
##vso[task.debug]loaded 0
##vso[task.debug]task result: Failed
##vso[task.issue type=error;]Input required: samplestring
##vso[task.complete result=Failed;]Input required: samplestring
```

As a fix, we can set the `samplestring` input and run the task again.

```
$env:INPUT_SAMPLESTRING="Human"
node index.js
##vso[task.debug]agent.workFolder=undefined
##vso[task.debug]loading inputs and endpoints
##vso[task.debug]loading INPUT_SAMPLESTRING
##vso[task.debug]loaded 1
##vso[task.debug]Agent.ProxyUrl=undefined
##vso[task.debug]Agent.CAInfo=undefined
##vso[task.debug]Agent.ClientCert=undefined
##vso[task.debug]Agent.SkipCertValidation=undefined
##vso[task.debug]samplestring=Human
Hello Human
```

This time, the task succeeded because `samplestring` was supplied and it correctly outputted "Hello Human!"

> **Tip**
>
> For information about various task runners and how to include the latest node version in the task.json, see **Node runner update guidance for Azure Pipelines task authors**.

2. Unit test your task scripts

Do unit tests to quickly test the task script, and not the external tools that it calls. Test all aspects of both success and failure paths.

1. Install test tools. We use Mocha as the test driver in this procedure.

```
npm install mocha --save-dev -g
npm install sync-request --save-dev
npm install @types/mocha --save-dev
```

2. Create a `tests` folder containing a `_suite.ts` file with the following contents:

TypeScript

```
import * as path from 'path';
import * as assert from 'assert';
import * as ttm from 'azure-pipelines-task-lib/mock-test';

describe('Sample task tests', function () {

    before( function() {

    });

    after(() => {

    });

    it('should succeed with simple inputs', function(done: Mocha.Done)
{
        // Add success test here
    });

    it('it should fail if tool returns 1', function(done: Mocha.Done) {
        // Add failure test here
    });
});
```

Tip

Your test folder should be located in the buildandreleasetask folder. If you get a sync-request error, you can work around it by adding sync-request to the buildandreleasetask folder with the command `npm i --save-dev sync-request`.

3. Create a `success.ts` file in your test directory with the following contents. This file creation simulates running the task and mocks all calls to outside methods.

```TypeScript
import ma = require('azure-pipelines-task-lib/mock-answer');
import tmrm = require('azure-pipelines-task-lib/mock-run');
import path = require('path');

let taskPath = path.join(__dirname, '..', 'index.js');
let tmr: tmrm.TaskMockRunner = new tmrm.TaskMockRunner(taskPath);

tmr.setInput('samplestring', 'human');

tmr.run();
```

   The success test validates that with the appropriate inputs, it succeeds with no errors or warnings and returns the correct output.

4. To run the task mock runner, add the following example success test to your `_suite.ts` file.

```TypeScript
     it('should succeed with simple inputs', function(done: Mocha.Done)
{
     this.timeout(1000);

     let tp: string = path.join(__dirname, 'success.js');
     let tr: ttm.MockTestRunner = new ttm.MockTestRunner(tp);

     // tr.run(); //current, old function.
     tr.runAsync().then(() => {
         console.log(tr.succeeded);
         assert.equal(tr.succeeded, true, 'should have succeeded');
         assert.equal(tr.warningIssues.length, 0, "should have no
warnings");
         assert.equal(tr.errorIssues.length, 0, "should have no
errors");
         console.log(tr.stdout);
         assert.equal(tr.stdout.indexOf('Hello human') >= 0, true,
"should display Hello human");
         done();
     }).catch((error) => {
         done(error); // Ensure the test case fails if there's an error
     });
 });
```

5. Create a `failure.ts` file in your test directory as your task mock runner with the following contents:

```
TypeScript
```

```typescript
import ma = require('azure-pipelines-task-lib/mock-answer');
import tmrm = require('azure-pipelines-task-lib/mock-run');
import path = require('path');

let taskPath = path.join(__dirname, '..', 'index.js');
let tmr: tmrm.TaskMockRunner = new tmrm.TaskMockRunner(taskPath);

tmr.setInput('samplestring', 'bad');

tmr.run();
```

   The failure test validates that when the tool gets bad or incomplete input, it fails in the expected way with helpful output.

6. To run the task mock runner, add the following code to your `_suite.ts` file.

```
TypeScript
```

```typescript
it('should fail if tool returns 1', function(done: Mocha.Done) {
    this.timeout(1000);

    const tp = path.join(__dirname, 'failure.js');
    const tr: ttm.MockTestRunner = new ttm.MockTestRunner(tp);

    tr.runAsync().then(() => {
        console.log(tr.succeeded);
        assert.equal(tr.succeeded, false, 'should have failed');
        assert.equal(tr.warningIssues.length, 0, 'should have no
warnings');
        assert.equal(tr.errorIssues.length, 1, 'should have 1 error
issue');
        assert.equal(tr.errorIssues[0], 'Bad input was given', 'error
issue output');
        assert.equal(tr.stdout.indexOf('Hello bad'), -1, 'Should not
display Hello bad');
        done();
    });
});
```

7. Run the tests.

```
```

```
tsc
```

```
mocha tests/_suite.js
```

Both tests should pass. If you want to run the tests with more verbose output (what you'd see in the build console), set the environment variable: `TASK_TEST_TRACE=1`.

```
$env:TASK_TEST_TRACE=1
```

3. Create the extension manifest file

The extension manifest contains all of the information about your extension. It includes links to your files, including your task folders and images folders. Ensure you created an images folder with extension-icon.png. The following example is an extension manifest that contains the build or release task.

Copy the following .json code and save it as your `vss-extension.json` file in your `home` directory.

Don't create this file in the buildandreleasetask folder.

javaScript

```
{
    "manifestVersion": 1,
    "id": "build-release-task",
    "name": "Fabrikam Build and Release Tools",
    "version": "0.0.1",
    "publisher": "fabrikam",
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "description": "Tools for building/releasing with Fabrikam. Includes one
build/release task.",
    "categories": [
        "Azure Pipelines"
    ],
    "icons": {
        "default": "images/extension-icon.png"
    },
    "files": [
        {
            "path": "buildandreleasetask"
        }
```

```
    ],
    "contributions": [
        {
            "id": "custom-build-release-task",
            "type": "ms.vss-distributed-task.task",
            "targets": [
                "ms.vss-distributed-task.tasks"
            ],
            "properties": {
                "name": "buildandreleasetask"
            }
        }
    ]
}
```

> **Note**
>
> Change the **publisher** to your publisher name. For more information, see Create a publisher.

Contributions

Expand table

Property	Description
`id`	Identifier of the contribution. Must be unique within the extension. Doesn't need to match the name of the build or release task. Typically the build or release task name is in the ID of the contribution.
`type`	Type of the contribution. Should be **ms.vss-distributed-task.task**.
`targets`	Contributions "targeted" by this contribution. Should be **ms.vss-distributed-task.tasks**.
`properties.name`	Name of the task. This name must match the folder name of the corresponding self-contained build or release pipeline task.

Files

Expand table

Property	Description
`path`	Path of the file or folder relative to the `home` directory.

> **Note**
>
> For more information about the extension manifest file, such as its properties and what they do, see the extension manifest reference.

4. Package your extension

Package all of your files together to get your extension into the Visual Studio Marketplace. All extensions are packaged as VSIX 2.0-compatible .vsix files. Microsoft provides a cross-platform command-line interface (CLI) to package your extension.

Once you have the tfx-cli, go to your extension's home directory, and run the following command:

```no-highlight
tfx extension create --manifest-globs vss-extension.json
```

> **Note**
>
> An extension or integration's version must be incremented on every update. When you're updating an existing extension, either update the version in the manifest or pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and saves the new version to your manifest. You must rev both the task version and extension version for an update to occur. `tfx extension create --manifest-globs vss-extension.json --rev-version` only updates the extension version and not the task version. For more information, see Build Task in GitHub.

Once your packaged extension is in a .vsix file, you're ready to publish your extension to the Marketplace.

5. Publish your extension

To publish your extension, you first create your publisher, then upload your extension, and finally share it.

Create your publisher

All extensions, including extensions from Microsoft, are identified as being provided by a publisher. If you aren't already a member of an existing publisher, you create one.

1. Sign in to the Visual Studio Marketplace Publishing Portal.
2. If you aren't already a member of an existing publisher, you're prompted to create a publisher. If you're not prompted to create a publisher, scroll down to the bottom of the page and select **Publish extensions** under **Related Sites**.

    - Specify an identifier for your publisher, for example: `mycompany-myteam`.
        - This identifier is used as the value for the `publisher` attribute in your extensions' manifest file.
    - Specify a display name for your publisher, for example: `My Team`.

3. Review the Marketplace Publisher Agreement and select **Create**.

Your publisher is defined. In a future release, you can grant permissions to view and manage your publisher's extensions. It's easier and more secure to publish extensions under a common publisher, without the need to share a set of credentials across users.

Upload your extension

Find the **Upload new extension** button, go to your packaged .vsix file, and select **Upload**.

1. You can also upload your extension via the command line interface (CLI) by using the `tfx extension publish` command instead of `tfx extension create` to package and publish your extension in one step. You can optionally use `--share-with` to share your extension with one or more accounts after it gets published.

    ```no-highlight
    tfx extension publish --manifest-globs your-manifest.json --share-with yourOrganization
    ```

2. Create a personal access token (PAT).

    - Select the "Marketplace (publish)" scope. This scope limits the token to only being able to publish extensions to the Marketplace.

Share your extension

Now that you uploaded your extension, it's in the Marketplace, but no one can see it. Share it with your organization so that you can install and test it.

Right-select your extension and select **Share**, and enter your organization information. You can share it with other accounts that you want to have access to your extension, too.

> **ⓘ Important**
>
> Publishers must be verified to share extensions publicly. For more information, see Package/Publish/Install.

Now that your extension is shared in the Marketplace, anyone who wants to use it must install it.

6. Create a build and release pipeline to publish the extension to Marketplace

To maintain the custom task on the Marketplace, create a build and release pipeline on Azure DevOps.

Prerequisites

Software/tool	Information
Azure DevOps project	Create a project.
Azure DevOps Extension Tasks extension	Install for free, Azure DevOps Extension Tasks in your organization.
Pipeline library variable group	Create a pipeline library variable group to hold the variables used by the pipeline. For more information, see Add and use variable groups. You can make variable groups from the Azure DevOps Library tab or through the CLI. Use the variables within this group in your pipeline. Also, declare the following variables in the variable group: • `publisherId`: ID of your marketplace publisher • `extensionId`: ID of your extension, as declared in the vss-extension.json file

- `extensionName`: Name of your extension, as declared in the vss-extension.json file
- `artifactName`: Name of the artifact being created for the VSIX file

Service connection

Create a new Marketplace service connection and grant access permissions for all pipelines.

YAML pipeline

Use the following example to create a new pipeline with YAML. For more information, see Create your first pipeline and YAML schema.

YAML

```yaml
trigger:
- main
pool:
vmImage: "ubuntu-latest"
variables:
- group: variable-group # Rename to whatever you named your variable group
in the prerequisite stage of step 6
stages:
- stage: Run_and_publish_unit_tests
jobs:
- job:
steps:
- task: TfxInstaller@4
inputs:
version: "v0.x"
- task: Npm@1
inputs:
command: 'install'
workingDir: '/TaskDirectory' # Update to the name of the directory of your
task
- task: Bash@3
displayName: Compile Javascript
inputs:
targetType: "inline"
script: |
cd TaskDirectory # Update to the name of the directory of your task
tsc
- task: Npm@1
inputs:
command: 'custom'
workingDir: '/TestsDirectory' # Update to the name of the directory of your
task's tests
customCommand: 'testScript' # See the definition in the explanation section
below - it may be called test
- task: PublishTestResults@2
inputs:
testResultsFormat: 'JUnit'
```

```
testResultsFiles: '**/ResultsFile.xml'
- stage: Package_extension_and_publish_build_artifacts
jobs:
- job:
steps:
- task: TfxInstaller@4
inputs:
version: "0.x"
- task: Npm@1
inputs:
command: 'install'
workingDir: '/TaskDirectory' # Update to the name of the directory of your
task
- task: Bash@3
displayName: Compile Javascript
inputs:
targetType: "inline"
script: |
cd TaskDirectory # Update to the name of the directory of your task
tsc
- task: QueryAzureDevOpsExtensionVersion@4
name: QueryVersion
inputs:
connectTo: 'VsTeam'
connectedServiceName: 'ServiceConnection' # Change to whatever you named the
service connection
publisherId: '$(PublisherID)'
extensionId: '$(ExtensionID)'
versionAction: 'Patch'
- task: PackageAzureDevOpsExtension@4
inputs:
rootFolder: '$(System.DefaultWorkingDirectory)'
publisherId: '$(PublisherID)'
extensionId: '$(ExtensionID)'
extensionName: '$(ExtensionName)'
extensionVersion: '$(QueryVersion.Extension.Version)'
updateTasksVersion: true
updateTasksVersionType: 'patch'
extensionVisibility: 'private' # Change to public if you're publishing to
the marketplace
extensionPricing: 'free'
- task: CopyFiles@2
displayName: "Copy Files to: $(Build.ArtifactStagingDirectory)"
inputs:
Contents: "**/*.vsix"
TargetFolder: "$(Build.ArtifactStagingDirectory)"
- task: PublishBuildArtifacts@1
inputs:
PathtoPublish: '$(Build.ArtifactStagingDirectory)'
ArtifactName: '$(ArtifactName)'
publishLocation: 'Container'
- stage: Download_build_artifacts_and_publish_the_extension
jobs:
- job:
steps:
```

```
- task: TfxInstaller@4
inputs:
version: "v0.x"
- task: DownloadBuildArtifacts@0
inputs:
buildType: "current"
downloadType: "single"
artifactName: "$(ArtifactName)"
downloadPath: "$(System.DefaultWorkingDirectory)"
- task: PublishAzureDevOpsExtension@4
inputs:
connectTo: 'VsTeam'
connectedServiceName: 'ServiceConnection' # Change to whatever you named the
service connection
fileType: 'vsix'
vsixFile: '$(PublisherID).$(ExtensionName)/$(PublisherID)..vsix'
publisherId: '$(PublisherID)'
extensionId: '$(ExtensionID)'
extensionName: '$(ExtensionName)'
updateTasksVersion: false
extensionVisibility: 'private' # Change to public if you're publishing to
the marketplace
extensionPricing: 'free'
```

For more information, see Specify events that trigger pipelines.

> ⓘ **Note**
>
> Each job uses a new user agent and requires dependencies to be installed.

Pipeline stages

The following section helps you understand how the pipeline stages work.

Stage 1: Run and publish unit tests

This stage runs unit tests and publishes test results to Azure DevOps.

To run unit tests, add a custom script to the package.json file like the following example.

JSON

```
"scripts": {
    "testScript": "mocha ./TestFile --reporter xunit --reporter-option
output=ResultsFile.xml"
},
```

1. Add "Use Node CLI for Azure DevOps (tfx-cli)" to install the tfx-cli onto your build agent.

2. Add the "npm" task with the "install" command and target the folder with the package.json file.

3. Add the "Bash" task to compile the TypeScript into JavaScript.

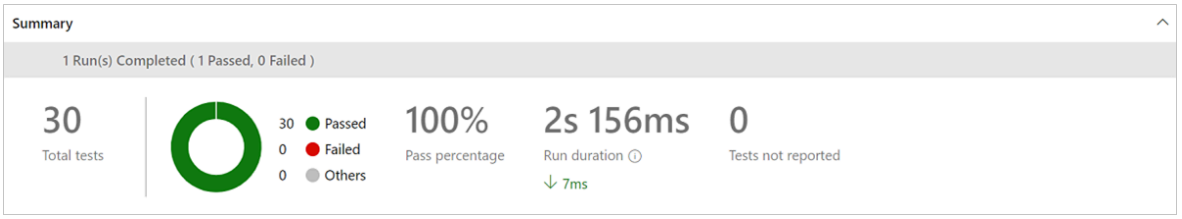
4. Add the "npm" task with the "custom" command, target the folder that contains the unit tests, and input `testScript` as the command. Use the following inputs:

   - Command: custom
   - Working folder that contains package.json: /TestsDirectory
   - Command and arguments: `testScript`

5. Add the "Publish Test Results" task. If you're using the Mocha XUnit reporter, ensure that the result format is "JUnit" and not "XUnit." Set the search folder to the root directory. Use the following inputs:

   - Test result format: JUnit
   - Test results files: **/ResultsFile.xml
   - Search folder: `$(System.DefaultWorkingDirectory)`

   After the test results get published, the output under the tests tab should look like the following example.



Stage 2: Package the extension and publish build artifacts

1. Add "Use Node CLI for Azure DevOps (tfx-cli)" to install the tfx-cli onto your build agent.

2. Add the "npm" task with the "install" command and target the folder with the package.json file.

3. Add the "Bash" task to compile the TypeScript into JavaScript.

4. To query the existing version, add the "Query Extension Version" task using the following inputs:

- Connect to: Visual Studio Marketplace
- Visual Studio Marketplace (Service connection): Service Connection
- Publisher ID: ID of your Visual Studio Marketplace publisher
- Extension ID: ID of your extension in the vss-extension.json file
- Increase version: Patch
- Output Variable: Task.Extension.Version

5. To package the extensions based on manifest Json, add the "Package Extension" task using the following inputs:

    - Root manifests folder: Points to root directory that contains manifest file. For example, $(System.DefaultWorkingDirectory) is the root directory
    - Manifest file: vss-extension.json
    - Publisher ID: ID of your Visual Studio Marketplace publisher
    - Extension ID: ID of your extension in the vss-extension.json file
    - Extension Name: Name of your extension in the vss-extension.json file
    - Extension Version: $(Task.Extension.Version)
    - Override tasks version: checked (true)
    - Override Type: Replace Only Patch (1.0.r)
    - Extension Visibility: If the extension is still in development, set the value to private. To release the extension to the public, set the value to public

6. To copy to published files, add the "Copy files" task using the following inputs:

    - Contents: All of the files to be copied for publishing them as an artifact
    - Target folder: The folder that the files get copied to
        - For example: $(Build.ArtifactStagingDirectory)

7. Add "Publish build artifacts" to publish the artifacts for use in other jobs or pipelines. Use the following inputs:

    - Path to publish: The path to the folder that contains the files that are being published
        - For example: $(Build.ArtifactStagingDirectory)
    - Artifact name: The name given to the artifact
    - Artifacts publish location: Choose "Azure Pipelines" to use the artifact in future jobs

Stage 3: Download build artifacts and publish the extension

1. Add "Use Node CLI for Azure DevOps (tfx-cli)" to install the tfx-cli onto your build agent.

2. To download the artifacts onto a new job, add the "Download build artifacts" task using the following inputs:

    - Download artifacts produced by: If you're downloading the artifact on a new job from the same pipeline, select "Current build." If you're downloading on a new pipeline, select "Specific build."
    - Download type: Choose "Specific artifact" to download all files that were published.
    - Artifact name: The published artifact's name.
    - Destination directory: The folder where the files should be downloaded.

3. The last task that you need is the "Publish Extension" task. Use the following inputs:

    - Connect to: Visual Studio Marketplace
    - Visual Studio Marketplace connection: ServiceConnection
    - Input file type: VSIX file
    - VSIX file: /Publisher.*.vsix
    - Publisher ID: ID of your Visual Studio Marketplace publisher
    - Extension ID: ID of your extension in the vss-extension.json file
    - Extension Name: Name of your extension in the vss-extension.json file
    - Extension visibility: Either private or public

Optional: Install and test your extension

Install an extension that is shared with you in just a few steps:

1. From your organization control panel (`https://dev.azure.com/{organization}/_admin`), go to the project collection administration page.
2. In the **Extensions** tab, find your extension in the "Extensions Shared With Me" group and select the extension link.
3. Install the extension.

If you can't see the **Extensions** tab, make sure you're in the control panel (the administration page at the project collection level, `https://dev.azure.com/{organization}/_admin`) and not the administration page for a project.

If you don't see the **Extensions** tab, then extensions aren't enabled for your organization. You can get early access to the extensions feature by joining the Visual Studio Partner Program.

To package and publish Azure DevOps Extensions to the Visual Studio Marketplace, you can download Azure DevOps Extension Tasks.

FAQs

See the following frequently asked questions (FAQs) about adding custom build or release tasks in extensions for Azure DevOps.

Q: How can I restrict Azure Pipelines commands usage for task?

You can restrict Azure Pipelines commands usage and variables, which get set by task. This action could be useful to prevent unrestricted access to variables/vso commands for custom scripts which task executes. We recommend that you set it up for new tasks. To apply, you might need to add the following statement to your task.json file:

JSON

```
  "restrictions": {
    "commands": {
      "mode": "restricted"
    },
    "settableVariables": {
      "allowed": ["variable1", "test*"]
    }
}
```

If `restricted` value is specified for `mode` - you can only execute the following commands by the task:

- `logdetail`
- `logissue`
- `complete`
- `setprogress`
- `setsecret`
- `setvariable`
- `debug`
- `settaskvariable`
- `prependpath`
- `publish`

The `settableVariables` restrictions allow you to pass in an allowlist of variables, which get set by `setvariable` or `prependpath` commands. It also allows basic regular expressions. For example, if your allowlist was: `['abc', 'test*']`, setting `abc`, `test`, or `test1` as variables with any value or prepending them to the path would succeed, but if you try to set a variable proxy it would warn. Empty list means that no variables get changed by task.

If either the `settableVariables` or `commands` key is omitted, relevant restriction isn't applied.

The restriction feature is available from 2.182.1 agent version.

Q: How is the cancellation signal handled by a task?

A: The pipeline agent sends `SIGINT` and `SIGTERM` signals to the relevant child process. There are no explicit means in the task library to process. For more information, see Agent jobs cancellation.

Q: How can I remove the task from project collection?

A: We don't support the automatic deletion of tasks. Automatic deletion isn't safe and breaks existing pipelines that already use such tasks. But, you can mark tasks as deprecated. To do so, bump the task version and mark the task as deprecated.

Q: How can I upgrade my custom task to the latest Node?

A: We recommend upgrading to the latest Node version. For example information, see Upgrading tasks to Node 20.

Microsoft Hosted agents and various Azure DevOps Server versions have different life cycles, leading to different Node runner versions being installed depending on where a task is running. To ensure compatibility across agents with different Node runner versions, the **task.json** file can include multiple execution sections. In the following example, Azure Pipeline agents with the Node 20 runner use it by default, while agent without it fall back to the Node 10 implementation.

Node.js

```
  "execution": {
    "Node10": {
      "target": "bash.js",
      "argumentFormat": ""
    },
```

```
    "Node20_1": {
      "target": "bash.js",
      "argumentFormat": ""
    }
```

To upgrade your tasks:

- To ensure your code behaves as expected, test your tasks on the various Node runner versions.
- In your task's execution section, update from `Node` or `Node10` to `Node16` or `Node20`.
- To support older server versions, you should leave the `Node`/`Node10` target. Older Azure DevOps Server versions might not have the latest Node runner version included.
- You can choose to share the entry point defined in the target or have targets optimized to the Node version used.

  Node.js

  ```
  "execution": {
     "Node10": {
       "target": "bash10.js",
       "argumentFormat": ""
  },
  "Node16": {
     "target": "bash16.js",
     "argumentFormat": ""
  },
  "Node20_1": {
     "target": "bash20.js",
     "argumentFormat": ""
  }
  ```

> **ⓘ Important**
>
> Not adding support for the Node 20 runner on your custom tasks will cause tasks to fail on agents installed from the `pipelines-agent-*` release feed.

Related articles

- Extension manifest reference
- Build/Release Task JSON Schema
- Build/Release Task Examples

ⓘ **Note:** The author created this article with assistance from AI. Learn more

Feedback

Was this page helpful? Yes No

Provide product feedback

Author a pipeline decorator

Article • 01/27/2023

Azure DevOps Services | Azure DevOps Server 2022

Pipeline decorators let you add steps to the beginning and end of every job. The process of authoring a pipeline decorator is different than adding steps to a single definition because it applies to all pipelines in an organization.

Suppose your organization requires running a virus scanner on all build outputs that could be released. Pipeline authors don't need to remember to add that step. We create a decorator that automatically injects the step. Our pipeline decorator injects a custom task that does virus scanning at the end of every pipeline job.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

1. Add contributions to an extension

The following example assumes you're familiar with the contribution models.

1. Create an extension. Once your extension gets created, you have a `vss-extension.json` file.
2. Add contributions to the `vss-extension.json` file for our new pipeline decorator.

vss-extension.json

JSON

```
{
    "manifestVersion": 1,
    "contributions": [
        {
            "id": "my-required-task",
            "type": "ms.azure-pipelines.pipeline-decorator",
            "targets": [
                "ms.azure-pipelines-agent-job.post-job-tasks"
            ],
            "properties": {
                "template": "my-decorator.yml"
            }
```

```
        }
    ],
    "files": [
        {
            "path": "my-decorator.yml",
            "addressable": true,
            "contentType": "text/plain"
        }
    ]
}
```

Contribution options

Let's take a look at the properties and what they're used for:

Property	Description
`id`	Contribution identifier. Must be unique among contributions in this extension.
`type`	Specifies that this contribution is a pipeline decorator. Must be the string `ms.azure-pipelines.pipeline-decorator`.
`targets`	Decorators can run before your job/specified task, after, or both. See the following table for available options.
`properties.template`	(Required) The template is a YAML file included in your extension, which defines the steps for your pipeline decorator. It's a relative path from the root of your extension folder.
`properties.targettask`	The target task ID used for `ms.azure-pipelines-agent-job.pre-task-tasks` or `ms.azure-pipelines-agent-job.post-task-tasks` targets. Must be GUID string like `89b8ac58-8cb7-4479-a362-1baaacc6c7ad`

Targets

Target	Description
`ms.azure-pipelines-agent-job.pre-job-tasks`	Run before other tasks in a classic build or YAML pipeline. Due to differences in how source code checkout happens, this target runs after checkout in a YAML pipeline but before checkout in a classic build pipeline.
`ms.azure-pipelines-agent-job.post-checkout-tasks`	Run after the last `checkout` task in a classic build or YAML pipeline.

Target	Description
`ms.azure-pipelines-agent-job.post-job-tasks`	Run after other tasks in a classic build or YAML pipeline.
`ms.azure-pipelines-agent-job.pre-task-tasks`	Run before specified task in a classic build or YAML pipeline.
`ms.azure-pipelines-agent-job.post-task-tasks`	Run after specified task in a classic build or YAML pipeline.
`ms.azure-release-pipelines-agent-job.pre-task-tasks`	Run before specified task in a classic RM pipeline.
`ms.azure-release-pipelines-agent-job.post-task-tasks`	Run after specified task in a classic RM pipeline.
`ms.azure-release-pipelines-agent-job.pre-job-tasks`	Run before other tasks in a classic RM pipeline.
`ms.azure-release-pipelines-agent-job.post-job-tasks`	Run after other tasks in a classic RM pipeline.

> ⓘ **Note**
>
> **Deployment jobs** in a YAML pipeline only support `ms.azure-pipelines-agent-job.pre-job-tasks` and `ms.azure-pipelines-agent-job.post-job-tasks` targets. **Jobs** support all YAML pipeline targets.

In this example, we use `ms.azure-pipelines-agent-job.post-job-tasks` because we want to run at the end of all build jobs.

This extension contributes a pipeline decorator. Next, we create a template YAML file to define the decorator's behavior.

2. Create a decorator YAML file

In the extension's properties, we chose the name "my-decorator.yml". Create that file in the root of your contribution. It holds the set of steps to run after each job. We start with a basic example and work up to the full task.

my-decorator.yml (initial version)

YAML

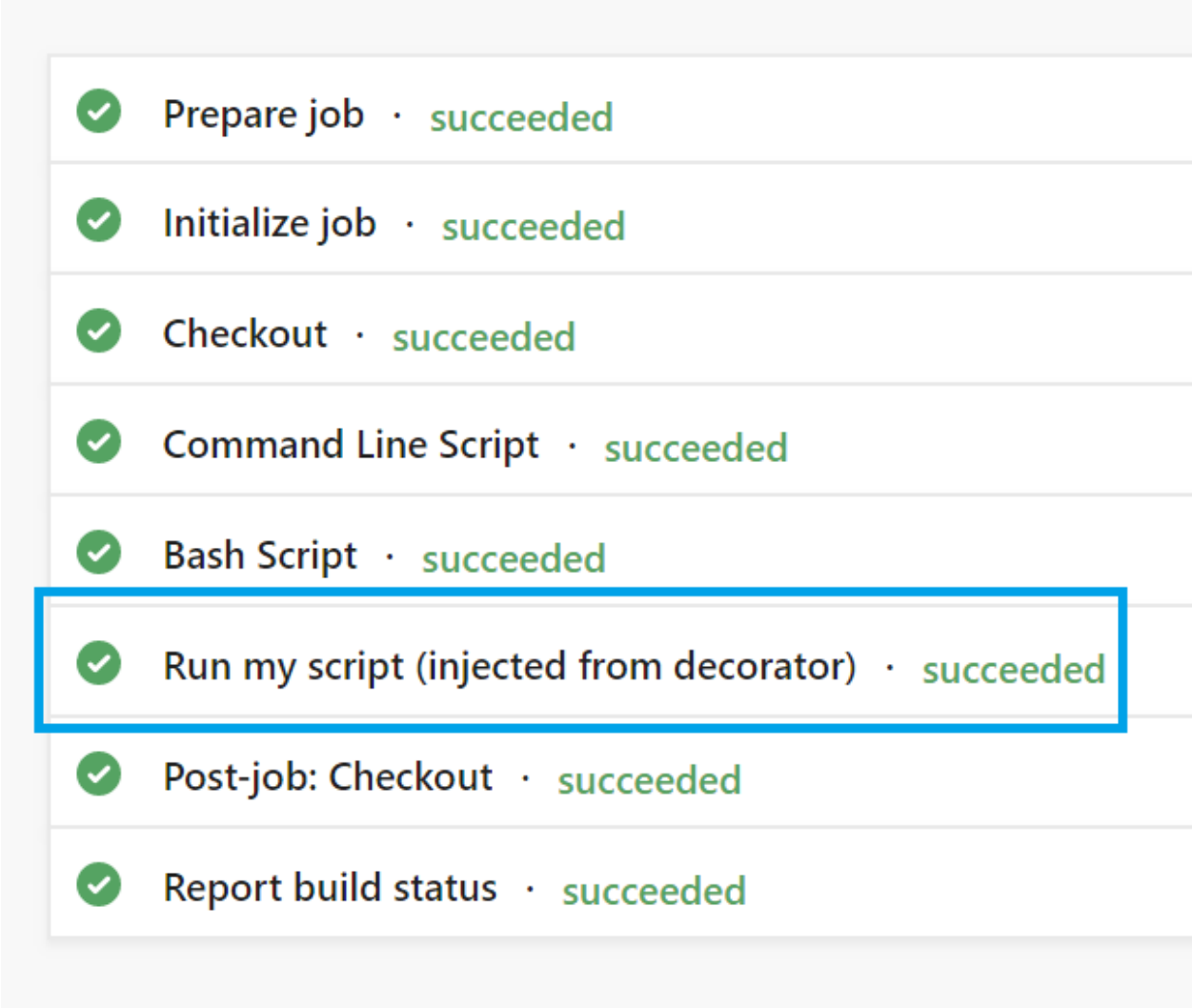
```yaml
steps:
- task: CmdLine@2
  displayName: 'Run my script (injected from decorator)'
  inputs:
    script: dir
```

3. Install the decorator

To add a pipeline decorator to your organization, you must install an extension. **Only private extensions can contribute pipeline decorators.** The extension must be authored and shared with your organization before it can be used.

Once the extension has been shared with your organization, search for the extension and install it.

Save the file, then build and install the extension. Create and run a basic pipeline. The decorator automatically injects our `dir` script at the end of every job. A pipeline run looks similar to the following example.



> ⓘ Note
>
> The decorator runs on every job in every pipeline in the organization. In later steps, we add logic to control when and how the decorator runs.

4. Inject conditions

In our example, we only need to run the virus scanner if the build outputs might be released to the public. Let's say that only builds from the default branch (typically `main`) are ever released. We should limit the decorator to jobs running against the default branch.

The updated file looks like this:

my-decorator.yml (revised version)

YAML

```
steps:
- ${{ if eq(resources.repositories['self'].ref,
resources.repositories['self'].defaultBranch) }}:
  - script: dir
    displayName: 'Run my script (injected from decorator)'
```

You can start to see the power of this extensibility point. Use the context of the current job to conditionally inject steps at runtime. Use YAML expressions to make decisions about what steps to inject and when. See pipeline decorator expression context for a full list of available data.

There's another condition we need to consider: what if the user already included the virus scanning step? We shouldn't waste time running it again. In this simple example, we'll pretend that any `script` task found in the job is running the virus scanner. (In a real implementation, you'd have a custom task to check for that instead.)

The script task's ID is `d9bafed4-0b18-4f58-968d-86655b4d2ce9`. If we see another script task, we shouldn't inject ours.

my-decorator.yml (final version)

YAML

```
steps:
- ${{ if and(eq(resources.repositories['self'].ref,
resources.repositories['self'].defaultBranch),
not(containsValue(job.steps.*.task.id, 'd9bafed4-0b18-4f58-968d-
86655b4d2ce9'))) }}:
  - script: dir
    displayName: 'Run my script (injected from decorator)'
```

5. Specify a target task

You can specify target task ID, and inject tasks before or after this target task. To specify target task, you can modify vss-extension.json manifest file like the following example.

vss-extension.json

JSON

```
{
    "contributions": [
        {
```

```
            "id": "my-required-task",
            "type": "ms.azure-pipelines.pipeline-decorator",
            "targets": [
                "ms.azure-pipelines-agent-job.pre-task-tasks",
                "ms.azure-pipelines-agent-job.post-task-tasks"
            ],
            "properties": {
                "template": "my-decorator.yml",
                "targettask": "target-task-id"
            }
        }
    ],
    ...
}
```

When you set up the 'targettask' property, you can specify ID of a target task. Tasks will be injected before/after all instances of specified target task.

Specify target task's inputs injection

You can specify a list of inputs of the target task that you want to inject as inputs to the injected task.

This feature is designed to work with custom pipeline tasks. It isn't intended to provide access to target pipeline task inputs via pipeline variables.

To get access to the target pipeline task inputs (inputs with the `target_` prefix), the injected pipeline task should use methods from the azure-pipelines-tasks-task-lib, and not the pipeline variables, for example `const inputString = tl.getInput('target_targetInput')`).

To do so, you can create your own custom pipeline task and use the target inputs there. If you need the functionality of one of the out-of-box tasks, like `CmdLine@2`, you can create a copy of the CmdLine@2 task and publish it with your decorator extension.

> **Note**
>
> This functionality is only available for tasks that are injected before or after the target task.

To specify this list of inputs, you can modify vss-extension.json manifest file like the following example.

vss-extension.json (injected task inputs version)

JSON

```
{
    "contributions": [
        {
            "id": "my-required-task",
            "type": "ms.azure-pipelines.pipeline-decorator",
            "targets": [
                "ms.azure-pipelines-agent-job.pre-task-tasks",
                "ms.azure-pipelines-agent-job.post-task-tasks"
            ],
            "properties": {
                "template": "my-decorator.yml",
                "targettask": "target-task-id",
                "targettaskinputs": ["target-task-input", "target-task-
second-input"]
            }
        }
    ],
    ...
}
```

By setting up of 'targettaskinputs' property, you can specify the list of inputs that are expected to inject. These inputs will be injected into the task with the prefix "`target_`" and will be available in the injected task like `target_target-task-input`.

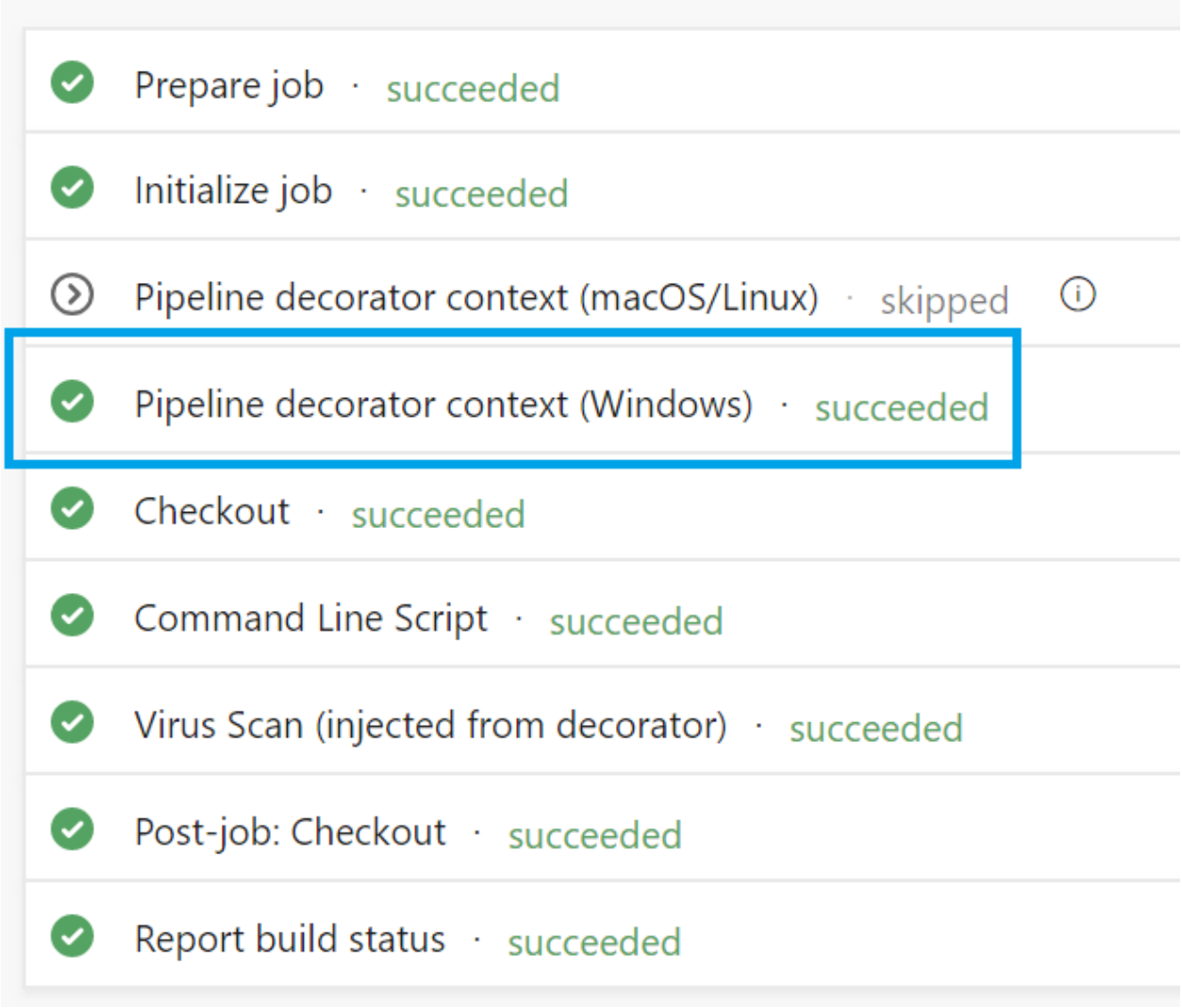
> **Note**
>
> Target task inputs that get secret values with variables or get them from other tasks won't be injected.

Debug

You might need to debug when you create your decorator. You also may want to see what data you have available in the context.

You can set the `system.debugContext` variable to `true` when you queue a pipeline. Then, look at the pipeline summary page.

You see something similar to the following image.



Select the task to see the logs, which show runtime values and that the context is available.

Related articles

- About YAML expression syntax
- Pipeline decorator expression context
- Develop a web extension
- Authentication guide

Extension samples

Article • 09/10/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Start developing your extension by working from a sample.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

The following Microsoft samples show the capabilities of the extension framework and how to contribute to various areas. Each sample illustrates one or more contributions. We limit the number of contributions for each sample to increase understanding of the extension framework. For source information, see the Azure DevOps extension samples repo.

Expand table

Sample	Contributions	Description
BreadcrumbService	Breadcrumb Service, Hub	Adds a breadcrumb service, which adds a "Sample Breadcrumb Item" to the sample hub. To see this item, go to the **Sample Hub** in the **Pipelines** hub group.
CodeEditorContribution	Code Editor	Adds a language definition and a JSON schema for the code editor.
Feature	Feature, Hub, Property Provider	Shows how to hook into the **Preview Features** panel under the user profile menu. Adds a simple hub that only shows when you turn on an "ABC" feature. You can toggle the feature on and off, per user or per organization.
Hub	Hub	Adds a hub named **Sample Hub** into the **Pipelines** hub group. The Sample Hub is on a project-level page, under the **Pipelines** navigation element.
Menu	Build Definition Menu Item	Adds a **Sample build definition menu item** to the **Builds** hub in the dropdown actions menu. The menu handler gets the current build definition from the context passed, makes a REST call, and then shows the result in a message box.

Sample	Contributions	Description
Panel	Panel Content, Hub	Applied within the **Hub** sample. Contains a toggle button along with **OK** and **Cancel** buttons. Can be used as a custom panel or dialog content.
Pivot	Web Tab	Adds a **Sample Pivot** tab to the organization or project collection home page, next to **Projects**, **My work items**, and **My pull requests**.
Pills	Pill Provider	Adds pills to the title of the Pipeline definition (Runs) page.
QueryParamsHandler	Event Subscription	Adds a service that loads on any page whenever a `showMyPanel` query parameter presents in the URL when any page gets loaded. The startup service shows the custom panel from the Panel sample, using an optional `myPanelTitle` query parameter as the panel title.
RepositoryActions	Menu Item	Adds a **Sample repository action** menu item to the repository picker in the header of code hub pages. If the `href` property shows, select the action to go to the given URL. If the `uri` property is provided, that code executes when you select the action.
RepositoryServiceHub	Hub	Adds a **Repository Information** hub to the **Code** hub group. Demonstrates how to interact with the `IVersionControlRepositoryService` to obtain basic information about a user's Git repository.
WorkItemFormGroup	Work Item Form Group	Adds a **Sample WorkItem Form Group** extension to the work item form to show how to interact with the `IWorkItemFormService` service and `IWorkItemNotificationListener`. Provides a UI to show case how to change field values using the form service and displaying work item form notification events.
WorkItemOpen	Hub	Adds a **Sample WorkItem Open** hub to the **Boards** hub group to show how to interact with the `IWorkItemFormNavigationService` service. Provides a UI for you to open an existing work item by ID, or open the work item form for a new work item by work item type. Either of these options open a dialog in the host frame.

DevLabs examples

Other open source examples that you might be interested in.

Expand table

Sample	Source	Contributions	Description
Team Calendar	GitHub	Hub, Event sources	Track events important to your team, view and manage days off, quickly see when sprints start and end, and more.
WSJF (Weighted Shortest Job First)	GitHub	Notification (work item), Context menu action (work item)	Auto calculates WSJF (weighted shortest job first) per work item and stores it in a work item field.
Cascading Lists	GitHub	Work Item Form	Define cascading behavior for picklists in work item form.
Retrospectives	GitHub	Hub	First-class experience for retrospectives and general feedback board scenarios. Collect feedback on your project milestones, organize and prioritize, and create and track actionable tasks, which can help your team improve over time.
Estimate	GitHub	Hub, Work item action menu	Play Planning Poker in Azure DevOps. Select work from an iteration, query, or your backlog, estimate the effort of those items with your team, and immediately update the work items.
Multivalue control	GitHub	Work item form	A work item form control, which allows selection of multiple values.
Azure DevOps Extension Tasks	GitHub	Build and release tasks	Azure Pipelines tasks for packaging and publishing Azure Devops and Visual Studio extensions to the Visual Studio Marketplace.

Get started

To get started as quickly as possible, use the seed project that contains the files required to build an extension using TypeScript. There's a grunt script to automate building, packaging, and publishing the extension.

Related articles

- Develop a web extension
- Use the Developer Formula Design System
- Use the Azure DevOps sample extensions repo
- View the contribution model
- Package and publish extensions

Feedback

Was this page helpful? Yes No

Provide product feedback

Vertical navigation guidance

Article • 10/04/2022

Azure DevOps Services

Vertical navigation brings with it changes that affect some extensions. These outcomes include support for extension icons along with changes to team context.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Team context

In traditional horizontal navigation, users could go to a project or team by selecting from a picker that's in the top left of the page header. This picker presented a list of recent teams and a way to browse for all teams. In the new vertical navigation, a user can only navigate into a project (and not into a team). This change was made to simplify the overall experience. But, it introduced a challenge for web extensions that rely on users' ability to switch teams using the traditional team picker in the page header.

`SDK.getWebContext()` is a client-side API provided by the SDK that provides information about the current organization, project, and team the user is operating in:

JSON

```
{
    "account": {
        "name": "Fabrikam",
        "id": "50e49b94-4bb6-40e7-8c85-a6d756d8a4d8"
    },
    "project": {
        "name": "Fabrikam Dev",
        "id": "049b1af2-fc87-4e50-95e4-151357f04b7f"
    },
    "team": {
        "name": "Ops Team",
        "id": "9b3a6b80-fe95-4c8c-8aa1-1e5d535d7af1"
    }
}
```

We don't recommend relying on `SDK.getWebContext().team`. Instead, follow the guidance below, based on the category your extension falls under.

Hub extensions that are team aware

If your extension needs to provide users a way to select a team, you can use the Teams REST API to get a list of teams for the current project. The following example shows how to call this API from your extension.

JavaScript

```
import { getClient } from "azure-devops-extension-api";
import { CoreRestClient } from "azure-devops-extension-api/Core";
import * as SDK from "azure-devops-extension-sdk";

private async getTeams() {
    const client = getClient(CoreRestClient);

    client.getTeams(SDK.getWebContext().project.id).then(
        function(teams) {
            console.log(teams);
        }
    );
}
```

For an example of an extension that provides a team picker control, see Team Calendar.

Pivots/Panels extensions that are in team aware hubs like Backlogs and Dashboard

Your extension can check the *configuration* object passed to your contribution. This object has different properties depending on the contribution type and where the contribution is hosted. Example shows reading team from the *configuration* and falling back to reading team from *webContext* to support both new vertical navigation and older horizontal navigation in on-premise releases.

JavaScript

```
function getCurrentTeam() {
  let webContext = SDK.getWebContext();
  let configuration = SDK.getConfiguration();

  if ("team" in configuration) {
    return configuration.team;
  } else if ("team" in webContext) {
```

```
    return webContext.team;
  } else {
    return null; // should only happen if not in a project context
  }
}
```

Actions extensions that are in team aware hubs like Backlogs and Dashboard

Your extension can check the *actionContext* object passed to the callback invoked when a user selects the contributed menu item. Example shows reading team from the *actionContext*.

JavaScript

```
var menuContributionHandler = (function () {
        "use strict";
        return {
            // This is a callback that gets invoked when a user selects the
newly contributed menu item
            // The actionContext parameter contains team information.
            execute: function (actionContext) {
                if("team" in actionContext) {
                    alert(`Team. Id is ${actionContext.team.id}, Name is
${actionContext.team.name}`);
                }
            }
        };
    }());
```

Hub icon

You can optionally set an asset (like a .png or .jpg) as the icon for your hub. This icon appears next to the hub in the vertical navigation bar. It must be packaged with your extension.

> ⓘ **Note**
>
> These icons don't appear in horizontal navigation.

Complete the following steps to set an icon for your hub.

1. Set the `iconAsset` property of the hub contribution to the fully qualified asset identifier, which follows the pattern: `{publisher-id}.{extension-id}/{asset-path}`.

2. Add an entry for this asset in the `includesata` contribution property.

3. Package the asset with your extension by listing it in the `files` property at the root of your manifest.

Example #1:

JSON

```
{
  "id": "my-extension",
  "publisherId": "my-publisher",
  ...
  "contributions": [
        {
            "id": "example-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-code-web.code-hub-group"
            ],
            "properties": {
                "name": "My Hub",
                "iconAsset": "my-publisher.my-extension/images/fabrikam-
logo.png",
                "includesData": {
                    "assets": [
                        "my-publisher.my-extension/images/fabrikam-logo.png"
                    ]
                }
            }
        }
  ],
 "files": [
     {
         "path": "images/fabrikam-logo.png",
         "addressable": true
     }
 ]
}
```

Example #2:

When themes like light versus dark get applied, you can specify the icons in your extension manifest as follows.

JSON

```
{
    "id": "hub",
    "type": "ms.vss-web.hub",
```

```
    "targets": [
        "ms.vss-work-web.work-hub-group"
    ],
    "properties": {
        "name": "Hub",
        "description": "Something",
        "uri": "pages/Hub.html",
        "icon": {
            "light": "img/hub-light.png",
            "dark": "img/hub-dark.png"
        }
    }
}
```

Work with URLs in extensions and integrations

Article • 02/14/2023

Azure DevOps Services

With the introduction of Azure DevOps, organizational resources and APIs are now accessible via either of the following URLs:

- `https://dev.azure.com/{organization}` (new)
- `https://{organization}.visualstudio.com` (legacy)

Regardless of when the organization was created, users, tools, and integrations can interact with organization-level REST APIs using either URL. As the developer of an extension, integration, or tool that interacts with Azure DevOps, learn how to best work with URLs made available to your code and form URLs when you call REST APIs.

For more information, see the REST API Reference.

Organization primary URL

Each organization has a designated **primary** URL that is either the new form or the legacy form. The primary URL is used by Azure DevOps for constructing URLs in certain scenarios (more details follow). The default primary URL for an organization is determined by when the organization was created, but can be changed by an administrator:

When the organization was created	Default primary URL
On or after 9/10/2018	New
Before 9/10/2018	Legacy

How the primary URL is used

The primary URL is the base URL for all URLs constructed by Azure DevOps in background jobs and other automated scenarios. See the following examples.

- URLs provided to Azure Pipelines tasks via environment variables (like `SYSTEM_TEAMFOUNDATIONCOLLECTIONURI`)
- URLs included in service hooks event payloads (like URLs in `resourceContainers`)

- URLs in email, Slack, Microsoft Teams, and similar notifications

For example, the following task snippet displays the organization URL provided to the task:

PowerShell

```
$orgUrl = $env:SYSTEM_TEAMFOUNDATIONCOLLECTIONURI
Write-Host $orgUrl
```

If this task is executed on an organization where the primary URL is the new URL form, the output is `https://dev.azure.com/{organization}`. The same task executed on an organization where the primary URL is the legacy URL form outputs `https://{organization}.visualstudio.com`.

It's therefore important that Azure Pipeline tasks and services that receive events from service hooks handle both URL forms.

URLs returned in REST APIs

Regardless of an organization's primary URL, URLs returned in the response to a REST API call use the same base URL as the requesting URL. This function ensures clients calling a REST API using a legacy URL continues to get back URLs in the same (legacy) form. For example, when the Projects REST API is called using a legacy URL, URLs in the response use the legacy form:

Request

HTTP

```
GET https://Fabrikam.visualstudio.com/_apis/projects/MyProject
```

Response

JavaScript

```
{
  "id": "ef4de40d-3d96-4b80-a320-cfafe038ae57",
  "name": "MyProject",
  "url": "https://Fabrikam.visualstudio.com/_apis/projects/MyProject"
}
```

Calling the same API using the new URL (`https://dev.azure.com/Fabrikam/_apis/projects/MyProject`) results in URLs returned in the new URL form.

Best practices

To ensure your extension, tool, or integration is resilient to changing organization URL forms and to possible future changes to the location (domain) of a REST API:

1. Assume the form of the organization URL can change over time.
2. Avoid parsing a URL to construct another URL.
3. Don't assume a particular REST API always resides on the same domain.
4. Avoid storing URLs in your service.
5. When possible, use Microsoft-provided .NET, TypeScript (web), Node.js, and Python client libraries with Azure DevOps.

How to get an organization's URL

With just the organization's name or ID, you can get its base URL using the global Resource Areas REST API (`https://dev.azure.com/_apis/resourceAreas`). This API doesn't require authentication. It also provides information about the location (URL) of the organization and the base URL for REST APIs, which can live on different domains.

A resource area is a group of related REST API resources and endpoints. Each resource area has a well-known identifier (see the following table). Each resource area has an organization-specific base URL that can be used to form URLs for APIs in that resource area. For example, the base URL for "build" REST APIs for the Fabrikam might be `https://dev.azure.com/Fabrikam`, but the base URL for "release management" REST APIs might be `https://vsrm.dev.azure.com/Fabrikam`.

> **Note**
>
> The Resource Areas REST API returns URLs for the organization based on that organization's designated primary URL.

With the organization's name

There are a few ways to get the base URL for an organization using its name.

HTTP

Request

Replace `{organizationName}` with organization's name, for example "Fabrikam". `79134C72-4A58-4B42-976C-04E7115F32BF` is the ID for the "core" resource area, which is where important resources like "projects" are.

```
GET https://dev.azure.com/_apis/resourceAreas/79134C72-4A58-4B42-976C-
04E7115F32BF
      ?accountName={organizationName}&api-version=5.0-preview.1
```

Response

JavaScript

```
{
    "id": "79134C72-4A58-4B42-976C-04E7115F32BF",
    "name": "Core",
    "locationUrl": "https://dev.azure.com/Fabrikam"
}
```

The `locationUrl` reflects the organization's base URL.

With the organization's ID

To get the URL for an organization using its GUID identifier, use the `hostId` query parameter in the previous examples (instead of `accountName`). For example:

```
GET https://dev.azure.com/_apis/resourceAreas/79134C72-4A58-4B42-976C-
04E7115F32BF?hostId={organizationId}&api-version=5.0-preview.1
```

How to get the base URL for a REST API

Starting from an organization's URL, you can use the Resource Areas REST API to look up the correct base URL for any REST API you need to call. This process ensures your

code is resilient to the location (domain) of a REST API changing in the future and avoids potentially brittle logic.

> ⓘ **Note**
>
> If you are using the Microsoft-provided .NET, TypeScript (web), Node.js, or Python client library, URL lookup is handled for you. For example, in .NET when you construct a `VssConnection` and call `GetClient<T>`, the returned client is properly bound to the correct base URL for the REST APIs called by this client.

If you aren't using a Microsoft-provided client library:

1. Use the following table to find the resource area ID for the REST API you need to call. The resource area name usually appears after `/_apis/` in the REST API route. For example, the `/_apis/release/definitions` REST API belongs to the `release` resource area, which has an ID of `efc2f575-36ef-48e9-b672-0c6fb4a48ac5`.

2. Call the organization-level Resource Areas REST API (`{organizationUrl}/_apis/resourceAreas/{resourceAreaId}?api-version=5.0-preview.1`) and pass the resource area ID. For example:

   ```
   GET https://dev.azure.com/Fabrikam/_apis/resourceAreas/efc2f575-36ef-48e9-b672-0c6fb4a48ac5?api-version=5.0-preview.1
   ```

3. Use the `locationUrl` field from the JSON response as the base URL for calling other REST APIs for this area. In this example, the base URL for Release Management REST APIs is `https://vsrm.dev.azure.com/Fabrikam`.

Like the global Resource Areas REST API described earlier, no credentials are required to call the organization-level Resource Areas REST API.

Example: Pipelines task calling an Azure Pipelines releases REST API

In this example, a build task needs to call the Azure Pipelines releases REST API. It forms the correct base URL for this REST API call by using the organization URL (provided in an environment variable) and the Resource Areas REST API.

> ⓘ **Note**

Resource area IDs are fixed and can be safely embedded in tasks and other logic.

PowerShell

```
$orgUrl = $env:SYSTEM_TEAMFOUNDATIONCOLLECTIONURI
$releaseManagementAreaId = "efc2f575-36ef-48e9-b672-0c6fb4a48ac5"

# Build the URL for calling the org-level Resource Areas REST API for the RM
APIs
$orgResourceAreasUrl = [string]::Format("{0}/_apis/resourceAreas/{1}?api-
preview=5.0-preview.1", $orgUrl, $releaseManagementAreaId)

# Do a GET on this URL (this returns an object with a "locationUrl" field)
$results = Invoke-RestMethod -Uri $orgResourceAreasUrl

# The "locationUrl" field reflects the correct base URL for RM REST API
calls
$rmUrl = $results.locationUrl

# Construct the URL to the release definitions REST API
$releaseDefinitionsUrl = [string]::Format("{0}/_apis/release/definitions?
api-preview=5.0-preview.1", $rmUrl)
```

Resource area IDs (reference)

This table shows the IDs for common resource areas. See the previous section for details on how to use this table.

> **Note**
>
> Resource area IDs are fixed and are consistent across all organizations in Azure DevOps Services.

Resource Area ID	Name
0d55247a-1c47-4462-9b1f-5e2125590ee6	account
5d6898bb-45ec-463f-95f9-54d49c71752e	build
79bea8f8-c898-4965-8c51-8bbc3966faa8	collection
79134c72-4a58-4b42-976c-04e7115f32bf	core
31c84e0a-3ece-48fd-a29d-100849af99ba	dashboard
a0848fa1-3593-4aec-949c-694c73f4c4ce	delegatedAuth

Resource Area ID	Name
6823169a-2419-4015-b2fd-6fd6f026ca00	discussion
a85b8835-c1a1-4aac-ae97-1c3d0ba72dbd	distributedtask
7bf94c77-0ce1-44e5-a0f3-263e4ebbf327	drop
6c2b0933-3600-42ae-bf8b-93d4f7e83594	extensionManagement
67349c8b-6425-42f2-97b6-0843cb037473	favorite
4e080c62-fa21-4fbc-8fef-2a10a2b38049	git
4e40f190-2e3f-4d9f-8331-c7788e833080	graph
68ddce18-2501-45f1-a17b-7931a9922690	memberEntitlementManagement
b3be7473-68ea-4a81-bfc7-9530baaa19ad	NuGet
4c83cfc1-f33a-477e-a789-29d38ffca52e	npm
45fb9450-a28d-476d-9b0f-fb4aedddff73	package
7ab4e64e-c4d8-4f50-ae73-5ef2e21642a5	packaging
2e0bf237-8973-4ec9-a581-9c3d679d1776	pipelines
fb13a388-40dd-4a04-b530-013a739c72ef	policy
8ccfef3d-2b87-4e99-8ccb-66e343d2daa8	profile
efc2f575-36ef-48e9-b672-0c6fb4a48ac5	release
57731fdf-7d72-4678-83de-f8b31266e429	reporting
ea48a0a1-269c-42d8-b8ad-ddc8fcdcf578	search
3b95fb80-fdda-4218-b60e-1052d070ae6b	test
c83eaf52-edf3-4034-ae11-17d38f25404c	testresults
8aa40520-446d-40e6-89f6-9c9f9ce44c48	tfvc
970aa69f-e316-4d78-b7b0-b7137e47a22c	user
5264459e-e5e0-4bd8-b118-0985e68a4ec5	wit
1d4f49f9-02b9-4e26-b826-2cdb6195f2a9	work
85f8c7b6-92fe-4ba6-8b6d-fbb67c809341	worktracking

Next steps

Make your extension or integration public

Data storage

Article • 01/29/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Azure DevOps extensions can store user preferences and complex data structures directly on Microsoft-provided infrastructure, which ensures your user's data is secure and backed up just like other organization and project data. It also means for simple data storage needs, you, as the extension provider, aren't required to set up, manage, or pay for third-party data storage services.

There are two methods to engage with the data storage service: through REST APIs or via a client service provided by Microsoft, which is part of the VSS SDK. We advise extension developers to utilize the provided client service APIs, as they offer a user-friendly encapsulation of the REST APIs.

> **Note**
>
> Looking for Azure DevOps REST APIs? See the latest Azure DevOps REST API reference.
>
> For information about .NET client libraries, see .NET client libraries for Azure DevOps.

What you can store

The service is designed to let you store and manage two different types of data:

- **Settings**: simple key-value settings (like user preferences)
- **Documents**: collections of similar complex objects (documents)

A collection is as an indexed container for documents. A document is a JSON blob that belongs to a collection. Other than a few reserved property names, you control and manage the schema of these documents.

How you can scope data

Settings and document collections can be scoped to either the:

- **Project Collection**: shared by all users of the project collection to which the extension is installed

- **User**: a single user of a project collection to which the extension is installed

Settings storage

The two main methods for managing settings are `getValue()` and `setValue()`:

- `getValue()` accepts a string key (along with other options like scope) and returns an IPromise. The resolved value of this promise is the value associated with the provided key.
- `setValue()` accepts a string key, a value, and other options such as scope, and returns an IPromise. The resolved value of this promise is the updated value of the setting.

Here's an example of how to set a value:

JavaScript

```
        private async initializeState(): Promise<void> {
        await SDK.ready();
        const accessToken = await SDK.getAccessToken();
        const extDataService = await SDK.getService<IExtensionDataService>
(CommonServiceIds.ExtensionDataService);
        this._dataManager = await
extDataService.getExtensionDataManager(SDK.getExtensionContext().id,
accessToken);

        this._dataManager.getValue<string>("test-id").then((data) => {
            this.setState({
                dataText: data,
                persistedText: data,
                ready: true
            });
        }, () => {
            this.setState({
                dataText: "",
                ready: true
            });
        });
    }
```

Here's an example of how to retrieve a setting value:

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Get value in user scope
        dataService.getValue("userScopedKey", {scopeType:
```

```
"User"}).then(function(value) {
            console.log("User scoped key value is " + value);
        });
    });
```

If `scopeType` isn't specified, the settings are stored at the project collection level and they're accessible to all users in that project collection using the extension. Here's an example of how to set a setting value at the project collection level:

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Set value (default is project collection scope)
        dataService.setValue("someKey", "abcd-efgh").then(function(value) {
            console.log("Key value is " + value);
        });
    });
```

Data (collections of documents) storage

To handle more complex data beyond key-value pairs, you can utilize the concept of documents to execute CRUD operations on your extension's data. A document is a JSON blob, enhanced with two special properties: ID and `__etag`. If they're crucial to an extension's data model, IDs can be user-defined, or if left unspecified, the system generates them. These IDs must be unique within a specific collection. Since a collection refers to a particular scope and instance of an extension, it implies that the same document ID can be reused across different collections.

The following document operations are available:

- Get a document
- Create a document
- Set a document (create or update)
- Update a document
- Delete a document

There's also a single operation that can be performed on a collection: Get all documents

Get a document by ID

Obtaining a document from a collection using its identifier is straightforward, like the following example:

JavaScript

```
    // Acquire data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Retrieve document by id
        dataService.getDocument("MyCollection",
"MyDocumentId").then(function(doc) {
            // Assuming document has a property named foo
            console.log("Doc foo: " + doc.foo);
        });
    });
```

This operation tries to fetch a document with the ID "MyDocumentId" from the "MyCollection" collection. In the absence of a provided scope, the service defaults to using the collection scoped to the entire instance of this extension. If either this collection or a document with the specified ID doesn't exist, a 404 error is returned, which the extension should handle. The returned document is a JSON object that includes all its properties, along with the special ID and `__etag` properties utilized by the data storage service.

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Get document by id
        dataService.getDocument("MyCollection",
"MyDocumentId").then(function(doc) {
            // Assuming document has a property named foo
            console.log("Doc foo: " + doc.foo);
        });
    });
```

This call attempts to retrieve a document with the ID "MyDocumentId," from the collection "MyCollection." Since no scope is provided, the collection that the service uses gets scoped to the default of the entire instance of this extension. If this collection doesn't exist or a document with that ID doesn't exist, then a 404 gets returned, which the extension should handle. The document that is returned is a JSON object containing all of its own properties, in addition to the special ID and `__etag` properties used by the data storage service.

Create a document

To create a new document, perform a call like the following example:

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Prepare document first
        var newDoc = {
            fullScreen: false,
            screenWidth: 500
        };

        dataService.createDocument("MyCollection",
newDoc).then(function(doc) {
            // Even if no ID was passed to createDocument, one gets
generated
            console.log("Doc id: " + doc.id);
        });
    });
```

If the collection with the name and scope provided, doesn't yet exist, it gets created dynamically before the document itself is created.

If the document provided contains an `id` property, that value gets used as the unique ID for the document. Please note that the provided `id` should be limited to 50 characters. If that field doesn't exist, a GUID gets generated by the service and included on the document that is returned when the promise is resolved.

If another document in the collection already exists with the same ID as the one provided on the document, the operation fails. If the desired behavior is create a new document if the ID doesn't exist, but modify the existing document if it does, then the `setDocument()` method should be used.

Set a document (update or create)

The `setDocument()` function carries out an "upsert" operation - it modifies an existing document if its ID is present and matches a document in the collection. If the ID is absent or doesn't correspond to any document in the collection, then a new document is added to the collection.

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Prepare document first
        var myDoc = {
            id: 1,
            fullScreen: false,
            screenWidth: 500
        };
```

```
        dataService.setDocument("MyCollection", myDoc).then(function(doc) {
            console.log("Doc id: " + doc.id);
        });
    });
```

Update a document

The `updateDocument` function requires that the document being altered already resides in the collection. An exception is thrown if no ID is supplied or if the provided ID doesn't correspond to any document in the collection.

Here's an example of how update is used:

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        var collection = "MyCollection";
        var docId = "1234-4567-8910";
        // Get document first
        dataService.getDocument(collection, docId, { scopeType: "User"
}).then(function(doc) {
            // Update the document
            doc.name = "John Doe";
            dataService.updateDocument(collection, doc, { scopeType: "User"
}).then(function(d) {
                // Check the new version
                console.log("Doc version: " + d.__etag);
            });
        });
    });
```

Delete a document

This function deletes the document with the provided ID from the provided collection. If the collection doesn't exist or the document doesn't exist, a 404 gets returned.

Here's an example usage:

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        var docId = "1234-4567-8910";
        // Delete document
```

```
        dataService.deleteDocument("MyCollection", docId).then(function() {
            console.log("Doc deleted");
        });
    });
```

Get all documents in a collection

The following example retrieves all documents from the "MyCollection" collection using the data service, and then logs the number of documents to the console:

JavaScript

```
    // Get data service
    SDK.getService(SDK.getContributionId()).then(function(dataService) {
        // Get all document under the collection
        dataService.getDocuments("MyCollection").then(function(docs) {
            console.log("There are " + docs.length + " in the collection.");
        });
    });
```

This call retrieves all documents in a scoped collection, with a limit of 100,000 documents. If the collection is nonexistent, it returns a 404 error.

Advanced

How settings get stored

This call encapsulates the `setDocument` client method, supplying it with multiple pieces of data. As stated before, settings are internally saved as documents. Therefore, a basic document is dynamically generated, where the document's ID is the key given in the `setValue()` method. The document has two more properties. The `value` property holds the value passed to the method, and the `revision` property is set to `-1`. While the `revision` property is elaborated more in the "Working with Documents" section, in the context of settings, setting `revision` to `-1` in the document signifies that we're not concerned with the versioning of this settings document.

Because settings are stored as documents, we need to provide a collection name, indicating where to store the document. To keep things simple, when working with the `setValue()`/`getValue()` methods, the collection name is always the special name `$settings`. The previous call issues a PUT Request at the following endpoint:

httprequest

```
GET
_apis/ExtensionManagement/InstalledExtensions/{publisherName}/{extensionName
}/Data/Scopes/User/Me/Collections/%24settings/Documents
```

The request payload is like the following example:

JSON

```
{
                "id": "myKey",
                "__etag": -1,
                "value": "myValue"
}
```

REST APIs

Assuming this snippet is executed after the value is set, you should see an alert message containing the text "The value is myValue." The getValue method is again a wrapper around the REST APIs, issuing a GET request to the following endpoint:

httprequest

```
GET
_apis/ExtensionManagement/InstalledExtensions/{publisherName}/{extensionName
}/Data/Scopes/User/Me/Collections/%24settings/Documents/myKey
```

etags

The `__etag` field is used by the Data Storage Service for document concurrency management. Before an update is saved, the service verifies that the `__etag` of the currently stored document matches the `__etag` of the updated document. If they match, the `__etag` is incremented and the updated document is returned to the caller. If they don't match, it indicates that the document to be updated is outdated, and an exception is thrown. The extension writer is responsible for handling this exception gracefully, either by retrieving the latest `__etag` of the document, merging the changes, and retrying the update, or by notifying the user.

For some types of documents, the level of concurrency provided might not be necessary, and a last-in-wins model might be more suitable. In such cases, while editing the document, input -1 as the `__etag` value to signify this functionality. The previously mentioned settings service employs this model for storing settings and preferences.

Feedback

Was this page helpful? Yes No

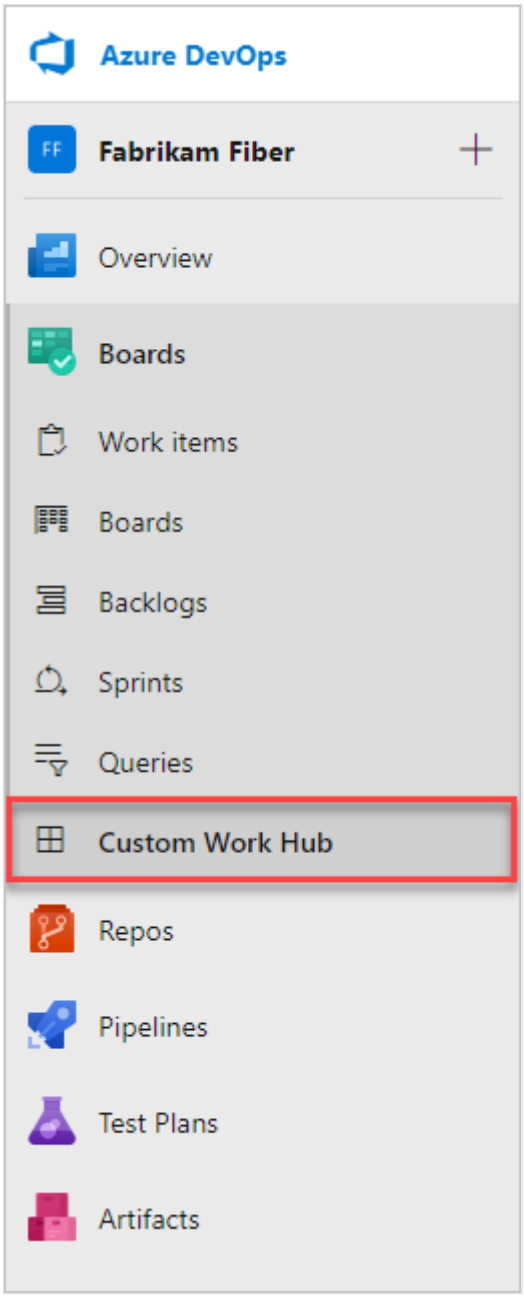
Provide product feedback

Add a hub

Article • 10/04/2022

Azure DevOps Services

In this article, we'll create a new hub that displays in Azure Boards after the *Sprints* and *Queries* hubs.



Structure of an extension

no-highlight

```
|--- README.md
|--- sdk
```

```
    |--- node_modules
    |--- scripts
        |--- SDK.js
|--- images
    |--- icon.png
|--- scripts                            // not used in this tutorial
|--- hello-world.html                  // html page to be used for your hub
|--- vss-extension.json                // extension's manifest
```

Get the client SDK: `SDK.js`

The core SDK script, SDK.js, enables web extensions to communicate to the host, Azure DevOps Services, frame. This script also initializes, notifies that the extension loaded, or gets context about the current page. Get the Client SDK `SDK.js` file and add it to your web app. Place it in the `home/sdk/scripts` folder.

Use the 'npm install' command via the command line (requires Node) to retrieve the SDK:

no-highlight

```
npm install azure-devops-extension-sdk
```

> **Note**
>
> For more information, see **Azure DevOps Web Extension SDK**.

Your hub page: `hello-world.html`

- Every hub displays a web page
- Check out the targetable hub groups in the extension points reference

Create a `hello-world.html` file in the `home` directory of your extension. Reference the SDK and call *init()* and *notifyLoadSucceeded()*.

HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
    <title>Hello World</title>
    <script src="sdk/scripts/SDK.js"></script>
</head>
```

```
<body>
    <script type="text/javascript">SDK.init();</script>
    <h1>Hello World</h1>
    <script type="text/javascript">SDK.notifyLoadSucceeded();</script>
</body>
</html>
```

Your extension's manifest file: `vss-extension.json`

- ***Every*** extension must have an extension manifest file
- Read the Extension manifest reference
- Find out more about the contribution points in Extensibility points

Create a json file (`vss-extension.json`, for example) in the `home` directory with the following contents:

JSON

```
{
    "manifestVersion": 1,
    "id": "sample-extension",
    "version": "0.1.0",
    "name": "My first sample extension",
    "description": "A sample Visual Studio Services extension.",
    "publisher": "fabrikamdev",
    "categories": ["Azure Boards"],
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
            }
        ],
    "icons": {
        "default": "images/logo.png"
     },
    "contributions": [
        {
            "id": "Fabrikam.HelloWorld",
            "type": "ms.vss-web.hub",
            "description": "Adds a 'Hello' hub to the Work hub group.",
            "targets": [
                "ms.vss-work-web.work-hub-group"
                ],
            "properties": {
                "name": "Hello",
                "order": 99,
                "uri": "hello-world.html"
            }
        }
```

```
        ],
        "scopes": [
            "vso.work"
        ],
        "files": [
            {
                "path": "hello-world.html", "addressable": true
            },
            {
                "path": "sdk/scripts", "addressable": true
            },
            {
                "path": "images/logo.png", "addressable": true
            }
        ]
    }
```

> ⓘ Note
>
> Change the **publisher** to your publisher name. To create a publisher, see **Package, publish, and install**.

Icons

The **icons** stanza specifies the path to your extension's icon in your manifest.

Add a square image titled `logo.png`, as shown in the extension manifest.

Contributions

The **contributions** stanza adds your contribution - the Hello hub - to your extension manifest.

For each contribution in your extension, the manifest defines the following:

- type of contribution, hub
- contribution target, the work hub group (check out all of the targetable hub groups),
- the properties specific to each type of contribution. A hub has the following properties.

Property	Description
name	Name of the hub.

Property	Description
order	Placement of the hub in the hub group.
uri	Path (relative to the extension base URI) of the page to surface as the hub.

Scopes

Include the scopes that your extension requires.

In this case, we need `vso.work` to access work items.

Files

The **files** stanza states the files that you want to include in your package - your HTML page, your scripts, the SDK script, and your logo.

Set `addressable` to `true` unless you include other files that don't need to be URL-addressable.

> ⓘ Note
>
> For more information about the **extension manifest file**, such as properties and function, check out the **extension manifest reference**.

Next steps

Package, publish, and install extensions

Related articles

- Testing and debugging extensions
- Extension manifest reference
- Azure DevOps web extension SDK
- Azure DevOps Formula Design System

Add a hub group

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

We'll create a hub group and add a hub to it. If you haven't already, create the Hello hub first, and then follow these steps to create the hub group.

1. Add a hub group to your app's manifest file in contributions, like this.

   JSON

   ```
   "contributions": [
       {
           "id": "sample-hub-group",
           "type": "ms.vss-web.hub-group",
           "description": "Adds a 'Samples' hub group at the project/team-
   level",
           "targets": [
               "ms.vss-web.project-hub-groups-collection"
           ],
           "properties": {
               "name": "Samples",
               "order": 100
           }
       },
    ]
   ```

   Look at the contribution targets reference to see the available hub groups that can be contributed to.

2. Change the hub contribution so that it's in the samples hub group that you just created. Just update the targets to the relative contribution ID of the hub group you just added.

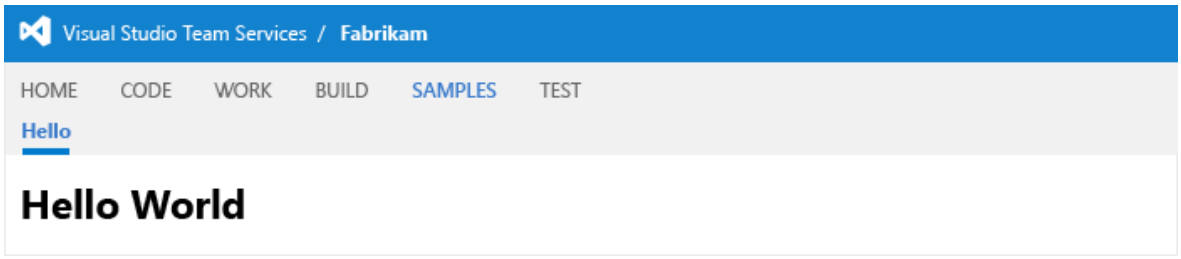
   JSON

   ```
       {
           "id": "Fabrikam.HelloWorld",
           "type": "ms.vss-web.hub",
           "description": "Adds a 'Hello' hub to the Work hub group.",
           "targets": [
               ".sample-hub-group"
           ],
           "properties": {
               "name": "Hello",
               "order": 99,
   ```

```
            "uri": "hello-world.html"
        }
    }
```

3. Install your extension.

   Now your hub appears under your Samples hub group.



Here's the complete extension manifest with Hello in the samples hub group.

JSON

```
{
    "namespace": "Fabrikam.myextension",
    "name": "My Extension",
    "description": "This is my first extension",
    "version": "1.0",
    "provider": {
        "name": "Fabrikam Fiber Inc"
    },
    "baseUri": "https://localhost:port",
    "icon": "images/logo.png",
    "links": {
        "info": "info.html",
        "support": "support.html",
        "termsOfService": "terms-of-service.html"
    },
    "contributions": {
        "vss.web#hubGroups.project": [
            {
                "id": "samples",
                "name": "Samples",
                "order":  30
            }
        ],
        "vss.web#hubs": [
            {
                "id": "myhub",
                "name": "Hello",
                "groupId": "samples",
                "uri": "hello-world.html"
            }
        ]
    }
}
```

Tip

Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Add a menu action

Article • 01/28/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

In this example, we add an action to the query context menu in the work item queries hub.

> **Tip**
>
> Check out our newest documentation on extension development using the Azure DevOps Extension SDK.

Prerequisites

- Develop a web extension.
- Create a web app for your action.

Update extension manifest file

Below is the code snippet that adds your action to the contributions section of your extension manifest.

JSON

```
...
    "contributions": [
        {
            "id": "myAction",
            "type": "ms.vss-web.action",
            "description": "Run in Hello hub action",
            "targets": [
                "ms.vss-work-web.work-item-query-menu"
            ],
            "properties": {
                "text": "Run in Hello hub",
                "title": "Run in Hello hub",
                "icon": "images/icon.png",
                "groupId": "actions",
                "uri": "action.html"
            }
        }
```

```
    ]
...
```

Properties

Expand table

Property	Description
text	Text that appears on the menu item.
title	Tooltip text that appears on the menu item.
icon	URL to an icon that appears on the menu item. Relative URLs are resolved using baseUri.
groupId	Determines where this menu item appears in relation to the others.
uri	URI to a page that registers the menu action handler (see below).
registeredObjectId	(Optional) Name of the registered menu action handler. Defaults to the contributor ID.

Learn about all of the places where you can add actions in Extensibility points.

Your HTML page

Your menu action is represented by a JavaScript script embedded in an HTML file. Save the following contents in a file and location that matches the reference to it in your extension's manifest file.

HTML

```
    <!DOCTYPE html>
    <html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Action Sample</title>
    </head>
    <body>
        <div>
            The end user doesn't see the content on this page.
            It is only in the background to handle the contributed menu item
being selected.
        </div>
    </body>
    </html>
```

Your JavaScript

The script below registers the handler object to handle the action, place it in the `head` section of the previous HTML page.

> We aliased `lib` to be `node_modules/azure-devops-extension-sdk/lib` in our `sdk-extension.json` manifest file.

TypeScript

```
<script src="lib/SDK.min.js"></script>
<script>
    SDK.init();

    // Use an IIFE to create an object that satisfies the
IContributedMenuSource contract
    var menuContributionHandler = (function () {
        "use strict";
        return {
            // This is a callback that gets invoked when a user selects the
newly contributed menu item
            // The actionContext parameter contains context data surrounding
the circumstances of this
            // action getting invoked.
            execute: function (actionContext) {
                alert("Hello, world");
            }
        };
    }());

    // Associate the menuContributionHandler object with the "myAction" menu
contribution from the manifest.
    SDK.register(SDK.getContributionId(), menuContributionHandler);
</script>
```

Tip

For more information, see Extensibility points, menus and toolbars, the Contribution model the Formula Design System, REST API reference, Extension samples, and resources in the Developer Community.

Next steps

Now that you've written your extension, the next steps are to Package, Publish, and Install your extension. You can also check out the documentation for Testing and

Debugging your extension.

- Package, publish, and install extensions
- Testing and debugging extensions

Feedback

Was this page helpful? Yes No

Provide product feedback

Create a service endpoint

Article • 03/25/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Service endpoints are a way for Azure DevOps to connect to external systems or services. They're a bundle of properties securely stored by Azure DevOps, which includes but isn't limited to the following properties:

- Service name
- Description
- Server URL
- Certificates or tokens
- User names and passwords

Extensions are then able to use the service endpoint to acquire the stored details to do the necessary operations on that service. Follow this guide to create a new service endpoint contribution and use it in your extension.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Task overview

You can develop a service endpoint by creating an example extension for Azure DevOps that includes the following items:

- A custom service endpoint with data sources, which enables a build task or dashboard widget to call a REST endpoint on the service/server defined by the endpoint.
- A build task, which defines two properties: The service endpoint & a picklist, which has values populated from the REST endpoint data source.

> **Note**
>
> When you create a service endpoints, it's at the project level, not the organization level.

The steps involved in completing this task are:

- 1. Create the extension manifest file
- 2. Create the build task pipeline, in the task.json file

> ⓘ **Note**
>
> This tutorial refers to the home directory for your project as "home".

Create the manifest file: `vss-extension.json`

The manifest file defines the custom endpoint and links to the task.json manifest for the build task.

In this article, the manifest file creation is separated into the following three parts:

- Create a service endpoint
  - Task overview
  - Create the manifest file: vss-extension.json
    - Create basic manifest file
    - Add the custom endpoint contribution
    - Add the build task contribution
  - Create the build task
    - task.json components
  - Authentication
  - Next steps
  - Related articles

Create basic manifest file

Create a json file (`vss-extension.json`, for example) in the `home` directory of your extension.

JSON

```
{
"manifestVersion": 1,
  "id": "service-endpoint-tutorial",
  "version": "0.1.1",
  "name": "Sample extension that leverages a service endpoint",
  "description": "A sample Azure DevOps extension which shows how to create
a custom endpoint and dynamic build task parameters taking value from a REST
API.",
```

```
  "publisher": "francistotten",
  "targets": [
    {
      "id": "Microsoft.VisualStudio.Services"
    }
  ],
  "files": [
    {
      "path": "BuildTaskFolder"
    }
  ]
}
```

> **Note**
>
> Update the `publisher` property. The `BuildTaskFolder` is the path where we'll eventually place our build task pipeline.

Add the custom endpoint contribution

Add the following `contributions` array underneath the `targets` array of the basic manifest content.

> **Important**
>
> Service connection parameters must be fetched by service connection ID.

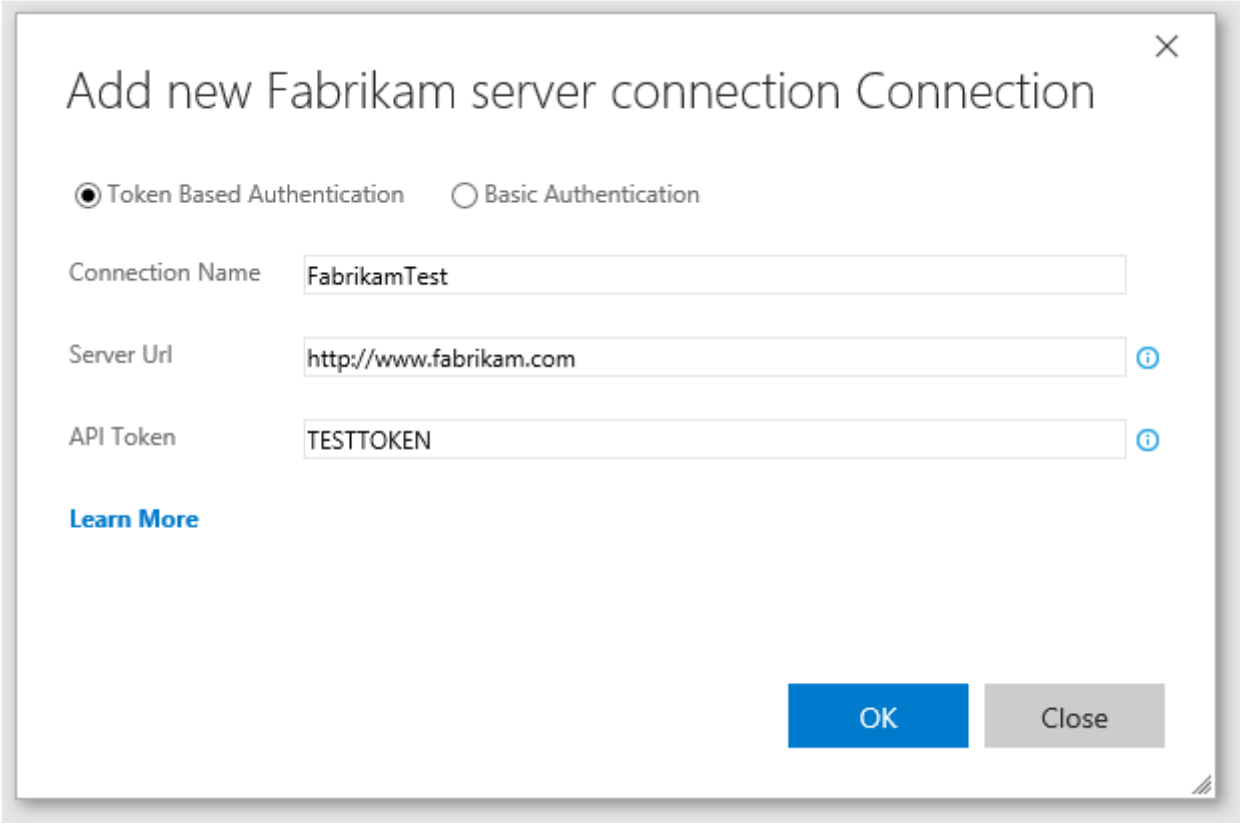
JSON

```
  "contributions": [
    {
      "id": "service-endpoint",
      "description": "Service endpoint type for Fabrikam connections",
      "type": "ms.vss-endpoint.service-endpoint-type",
      "targets": [ "ms.vss-endpoint.endpoint-types" ],
      "properties": {
        "name": "fabrikam",
        "displayName": "Fabrikam server connection",
        "url": {
          "displayName": "Server Url",
          "helpText": "Url for the Fabrikam server to connect to."
        },
        "dataSources": [
          {
            "name": "Fabrikam Projects",
            "endpointUrl": "{{endpoint.url}}api/projects/index",
            "resultSelector": "jsonpath:$[*].nm"
```

```
          }

        ],
        "authenticationSchemes": [
          {
            "type": "ms.vss-endpoint.endpoint-auth-scheme-token"
          },
          {
            "type": "ms.vss-endpoint.endpoint-auth-scheme-basic",
            "inputDescriptors": [
              {
                "id": "username",
                "name": "Username",
                "description": "Username",
                "inputMode": "textbox",
                "validation": {
                  "isRequired": false,
                  "dataType": "string"
                }
              },
              {
                "id": "password",
                "name": "Password",
                "description": "Password",
                "inputMode": "passwordbox",
                "isConfidential": true,
                "validation": {
                  "isRequired": false,
                  "dataType": "string"
                }
              }
            ]
          }

        ],
        "helpMarkDown": "<a href=\"url-to-documentation\" target=\"_blank\">
<b>Learn More</b></a>"
      }
    },
  ],
```

If you've successfully added the service contribution, you see the Fabrikam endpoint when you try to add a new service endpoint to your organization.

Create a service endpoint using the Fabrikam endpoint.



> **Tip**
>
> You can add inputDescriptors without authenticationSchemes. For more information, see InputDescriptor interface.

Add the build task contribution

Inside the `contributions` array from the previous step, add the following object to the end.

JSON

```
{
      "id": "build-task",
      "description": "Task with a dynamic property getting data from an
endpoint REST data source",
      "type": "ms.vss-distributed-task.task",
      "targets": [ "ms.vss-distributed-task.tasks" ],
      "properties": {
        "name": "BuildTaskFolder"
      }
    }
```

The dataSource endpoint URL gets computed from the URL of the endpoint or a fixed URL, and some other values. For this tutorial, this REST call returns nothing and is meant

to be replaced by any REST calls you wish to make to your service.

It's possible to use other parameters than the endpoint URL for the REST URL, for instance some endpoint properties. For instance, assuming that we had a property in the endpoint named subscriptionId, the REST URL could use it with the following syntax: $(endpoint.subscription).

Create the build task

The `task.json` file describes your build task.

> **Note**
>
> For more information, check out the following articles:
>
> - Build task reference to find the schema for the build task json file
> - Add a custom pipelines task extension
> - Integrate build task

Create a `task.json` file in your `BuildTaskFolder` directory, if you haven't created this folder yet, do so now.

JSON

```
{
  "id": "6557a6d2-4caf-4247-99ea-5131286a8753",
  "name": "build-task",
  "friendlyName": "Build Task that uses the service endpoint",
  "description": "Task with a dynamic property getting data from an endpoint
REST data source",
  "author": "francistotten",
  "helpMarkDown": "Replace with Markdown to show in help",
  "category": "Build",
  "visibility": [
    "Build",
    "Release"
  ],
  "demands": [],
  "version": {
    "Major": "0",
    "Minor": "1",
    "Patch": "1"
  },
  "minimumAgentVersion": "1.95.0",
  "instanceNameFormat": "Service Endpoint Build Task $(project)",
  "inputs": [
    {
```

```
      "name": "FabrikamService",
      "type": "connectedService:Fabrikam",
      "label": "Fabrikam service/server end point",
      "defaultValue": "",
      "required": true,
      "helpMarkDown": "Select the Fabrikam end point to use. If
needed,selecton 'manage', and add a new service endpoint of type 'Fabrikam
server connection'"
    },
    {
      "name": "project",
      "type": "pickList",
      "label": "Fabrikam Project",
      "required": true,
      "helpMarkDown": "Select the name of the Fabrikam Project to analyze.",
      "properties": {
        "EditableOptions": "True"
      }
    }
  ],
  "dataSourceBindings": [
    {
      "target": "project",
      "endpointId": "$(FabrikamService)",
      "dataSourceName": "Fabrikam Projects"
    }
  ],
  "execution": {
    "Node": {
      "target": "sample.js",
      "argumentFormat": ""
    },
    "PowerShell3": {
      "target": "sample.ps1"
    }
  }
}
```

task.json components

The `FabrikamService` input object

This field is the first of type connectedService:Fabrikam.connectedService expresses that it's an endpoint type, and that Fabrikam is the name of the object.

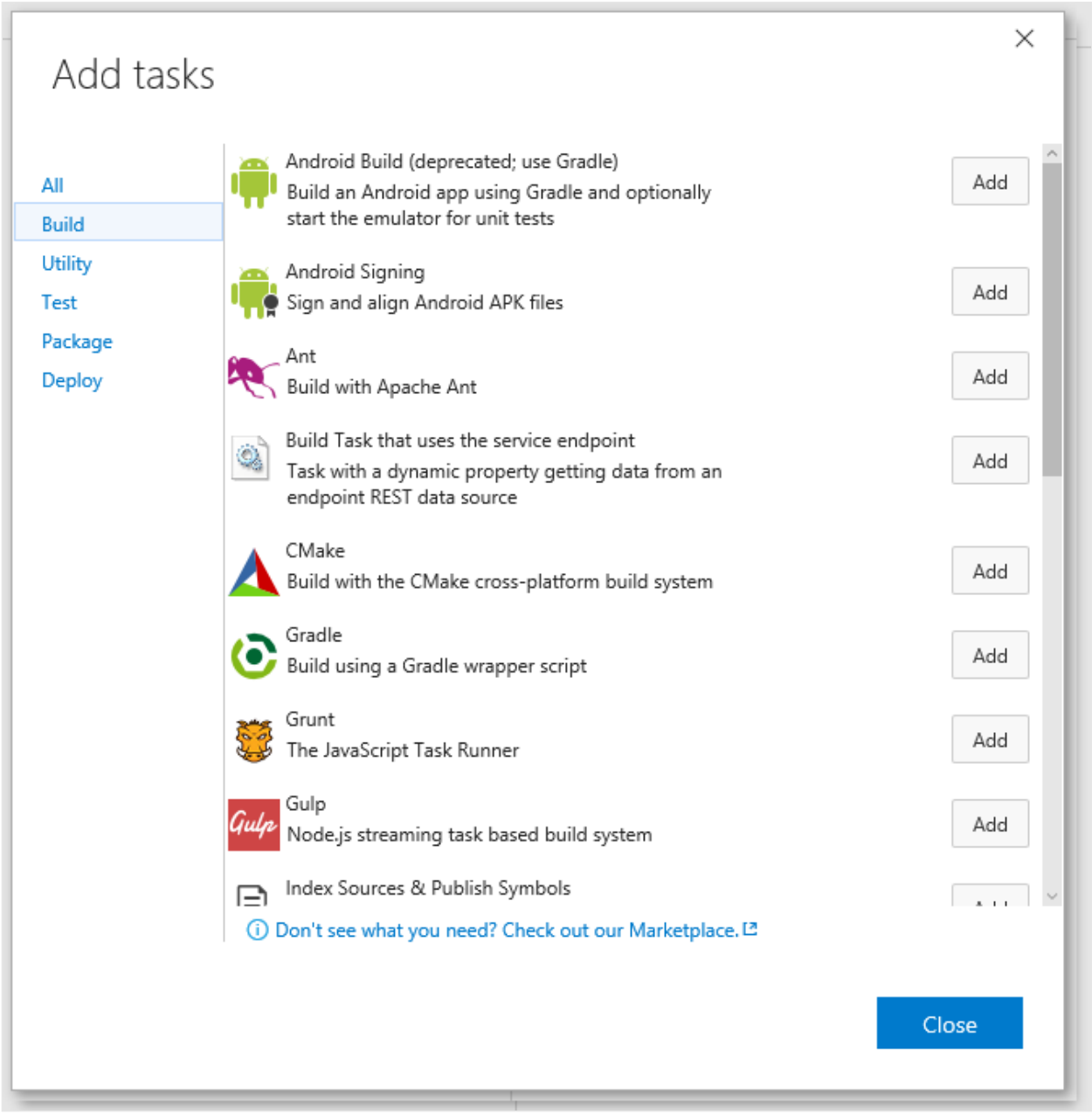
The `project` input object

This field is second. It's a picklist.

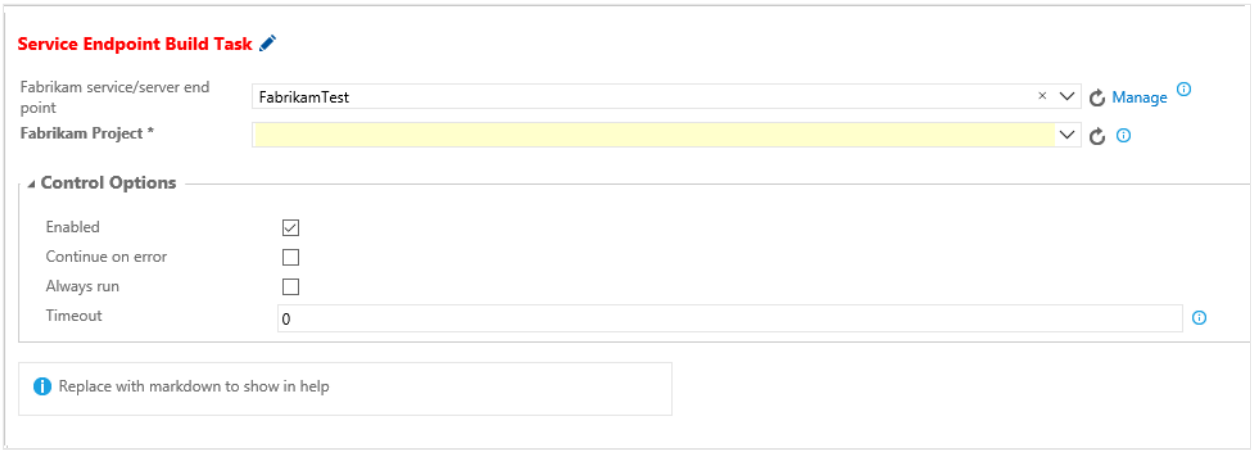
- This field is populated by a REST call.

- The values from the field "project" are taken from the "Projects" REST data source of the custom endpoint.
- Expressed in the `dataSourceBindings` array.
  - The target is the name of the build task field to be populated ("project").
  - The endpointId is the name of the build task field containing the custom endpoint type.
  - The REST call is chosen by the dataSourceName.

If you've added the Build Task successfully, you should now see the Build Task when you're adding tasks to a build pipeline.



Once you've added the Build Task to your pipeline, confirm that it can see the Fabrikam endpoint you created. The projects dropdown in this tutorial is blank since we aren't using a real service. Once you replace Fabrikam with your service, replace the Projects call with your own REST API call to use dynamic data inside your build task.



Authentication

The authentication scheme in a service endpoint determines the credentials that would be used to connect to the external service. For more information and to see the following authentication schemes, see the authentication schemes documentation.

- Basic authentication
- Token-based authentication
- Certificate-based authentication
- No authentication

Next steps

Package, publish, and install extensions

Related articles

- Test and debug extensions
- Develop a web extension
- Add a pipeline decorator

Feedback

Was this page helpful? Yes No

Provide product feedback

Create a custom consumer for service hooks

Article • 02/20/2023

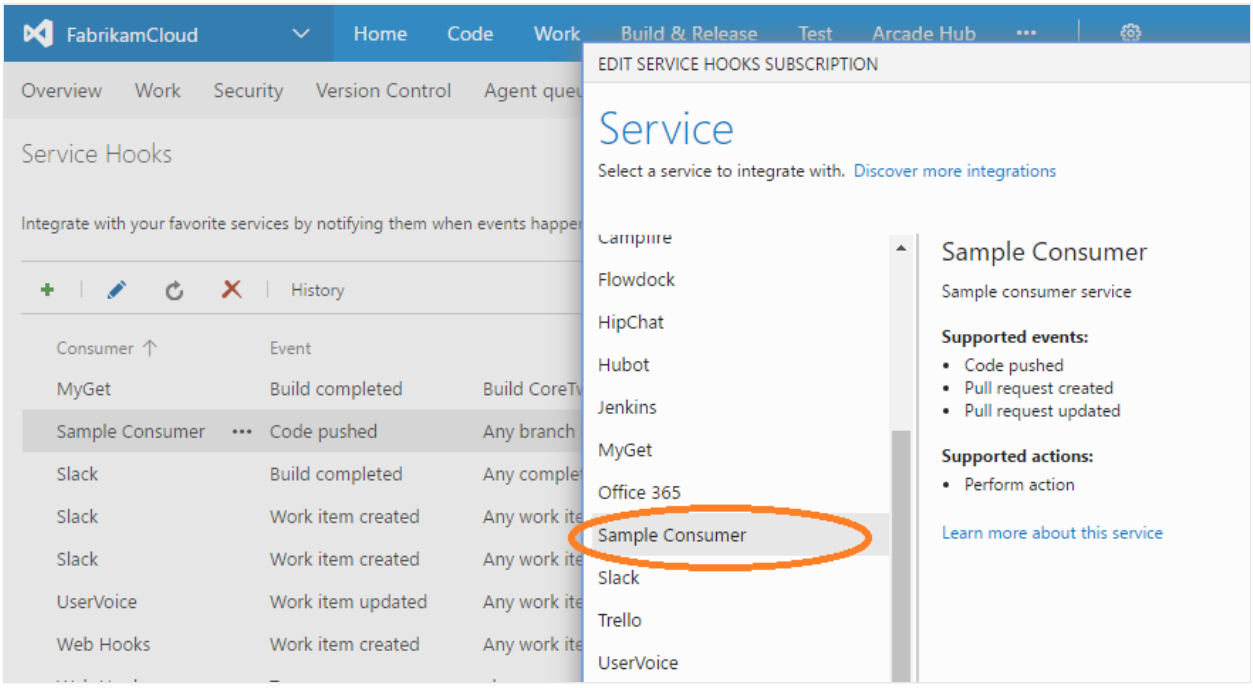
Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

With service hooks, you can notify third-party systems about events that occur in your project. You can use a custom consumer to send an HTTP message to the endpoint that's defined in the extension's manifest.

This article walks through developing an extension that implements a **sample consumer service**, which includes the following events and actions.

- Supported events that trigger the following actions:
  - Code pushed
  - Pull request created
  - Pull request updated
- Supported actions to take when events occur:
  - Do action (Send HTTP message)

> **Note**
>
> In this article, we refer to the home directory for your project as "home".

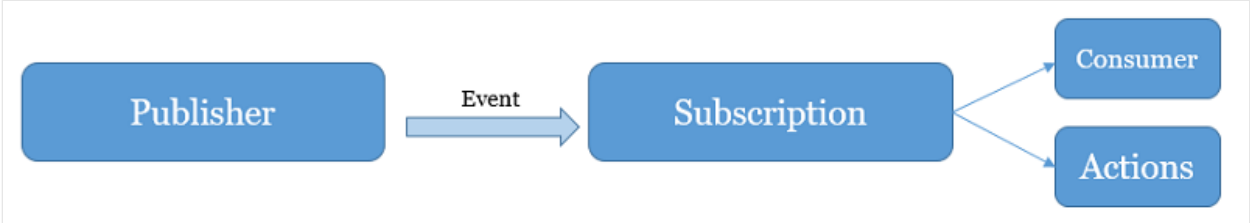


For more information, see the Extension example GitHub repo. For a list of all supported events you can use as triggers for your custom consumer extension, see List of event types.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

How service hooks work

Service hook **publishers** define a set of *events*. **Subscriptions** listen and wait for the *events* and define **actions** for when the event is triggered.



This is a general description of how all our service hook implementations work. For our case, we specify our consumer defined by an extension, as well as the specified action for when an event occurs.

Create the extension

1. See how to create your extension from scratch.
2. Add the specific contribution for custom consumer implementation to your basic manifest file. See the following example of how your manifest should look after you add the contribution.

JSON

```
{
    "manifestVersion": 1,
    "id": "samples-service-hooks-consumer",
    "version": "0.1.2",
    "name": "Service Hooks Sample",
    "description": "A simple extension that demonstrates how to contribute a
consumer service into service hooks.",
    "publisher": "fabrikam",
    "public": false,
    "icons": {
```

```
        "default": "images/logo.png"
    },
    "scopes": [],
    "files": [
        {
            "path": "images",
            "addressable": true
        }
    ],
    "content": {
        "details": {
            "path": "readme.md"
        }
    },
    "categories": [
        "Developer samples"
    ],
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "contributions": [
        {
            "id": "consumer",
            "type": "ms.vss-servicehooks.consumer",
            "targets": [
                "ms.vss-servicehooks.consumers"
            ],
            "properties": {
                "id": "consumer",
                "name": "Sample Consumer",
                "description": "Sample consumer service",
                "informationUrl": "https://aka.ms/vsoextensions",
                "inputDescriptors": [
                    {
                        "id": "url",
                        "isRequired": true,
                        "name": "URL",
                        "description": "URL to post event payload to",
                        "inputMode": "textbox"
                    }
                ],
                "actions": [
                    {
                        "id": "performAction",
                        "name": "Perform action",
                        "description": "Posts a standard event payload",
                        "supportedEventTypes": [
                            "git.push",
                            "git.pullrequest.created",
                            "git.pullrequest.updated"
                        ],
                        "publishEvent": {
                            "url": "{{{url}}}",
```

```
                        "resourceDetailsToSend": "all",
                        "messagesToSend": "all",
                        "detailedMessagesToSend": "all"
                    }
                }
            ]
        }
    }
  ]
}
```

> **Note**
>
> Remember to update the `publisher` property.

For each contribution in your extension, the manifest defines the following items.

- Type of contribution - consumer service (ms.vss-servicehooks.consumer) in this case
- Contribution target - consumer services (ms.vss-servicehooks.consumers) in this case
- Properties that are specific to each type of contribution

Consumers have the following properties.

Property	Description
id	Unique ID for your consumer service.
name	Name of the custom consumer, which is visible during service hook subscription creation.
description	Describes your consumer service.
informationUrl	Find more info about your extension.
inputDescriptors	Inputs to be used by users that are creating subscriptions with the consumer service.
actions	Describes the actions to take and which events trigger those actions.

Actions for your consumer have the following properties:

Property	Description
id	ID for your action service.

Property	Description
name	Name of the action.
description	Detailed description of the action.
supportedEventTypes	Array of trigger types for which this action can be used. For more information, see List of event types.
publishEvent.url	URL where HTTP message gets sent to. It can be templated by values provided by inputDescriptors. Their actual values get defined by the user when the subscription gets created.

3. Deploy your extension to your Azure DevOps organization and test it.

Next steps

Package, publish, and install extensions

Related articles

- Service hook consumers
- Available services for service hooks
- Create a service hook subscription programmatically
- Test and debug extensions
- Extension example GitHub repo
- List of event types

Create a pull request status server with Node.js

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The pull request (PR) workflow provides developers with an opportunity to get feedback on their code from peers as well as from automated tools. 3rd party tools and services can participate in the PR workflow by using the PR Status API. This article guides you through the process of creating a status server to validate PRs in an Azure DevOps Services Git repository. For more information about PR status, see Customize and extend pull request workflows with pull request status.

Prerequisites

- An organization in Azure DevOps with a Git repo. If you don't have an organization, sign up to upload and share code in free unlimited private Git repositories.
- Install VS Code or other code editor of your choice. The instructions in this guide use VS Code but the steps in other code editors are similar.

Install Node.js

To install Node.js, download the LTS release appropriate for your platform. The download contains an installer, which you can run to install the Node.js runtime on your local machine. When installing Node.js, be sure to keep the npm package manager portion of the install, which is selected by default.

Create a basic web server using Express

The steps in this section use Express, which is a lightweight web framework for Node.js that provides a number of HTTP utility methods that simplify creating a web server. This framework provides you with the basic functions needed to listen to PR events.

1. From the command line, create a new project folder for your web server.

```
```

```
mkdir pr-server
cd pr-server
```

2. Use the `npm init` command to create a new `package.json` file for the project.

```

npm init
```

   Press Enter to accept the defaults for all of the options except the entry point. Change it to `app.js`

```

entry point: (index.js) app.js
```

3. Install Express in the pr-server directory using the following command. This installs Express and saves it to the dependencies list.

```

npm install express
```

4. Create a simple Express app to build upon for the PR status server. The following steps are based on the Express Hello world example. Open the project folder in VS Code by running the following command from the `pr-server` folder.

```

code .
```

5. Create a new file `(Ctrl + N)` and paste in the following sample code.

JavaScript

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
res.send('Hello World!')
})

app.listen(3000, function () {
```

```
console.log('Example app listening on port 3000!')
})
```

6. Save the file as `app.js`.

7. Run the basic web server using the following command:

```
node app.js
```

   Verify the server is running by browsing to `http://localhost:3000/`.

Listen for HTTP POST requests

The web server is going to receive `POST` requests from Azure DevOps Services, so you need to handle those requests in your server.

1. At the end of the `app.js` file, add the following code, and save the file.

```
JavaScript

app.post('/', function (req, res) {
    res.send('Received the POST')
})
```

2. Re-run your web server using the following command:

```
node app.js
```

Configure a service hook for PR events

Service hooks are an Azure DevOps Services feature that can alert external services when certain events occur. For this sample, you'll want to set up two service hooks for PR events, so the status server can be notified. The first will be for the **Pull request created** event and the second will be for the **Pull request updated** event.

In order to receive the service hook notifications, you'll need to expose a port to the public internet. The ngrok utility is very useful for doing this in a development environment.

1. Download and unzip the appropriate ngrok release for your platform.

2. Use ngrok to start listening on the same port as your sample server - port 3000. Run the following command in a new command window.

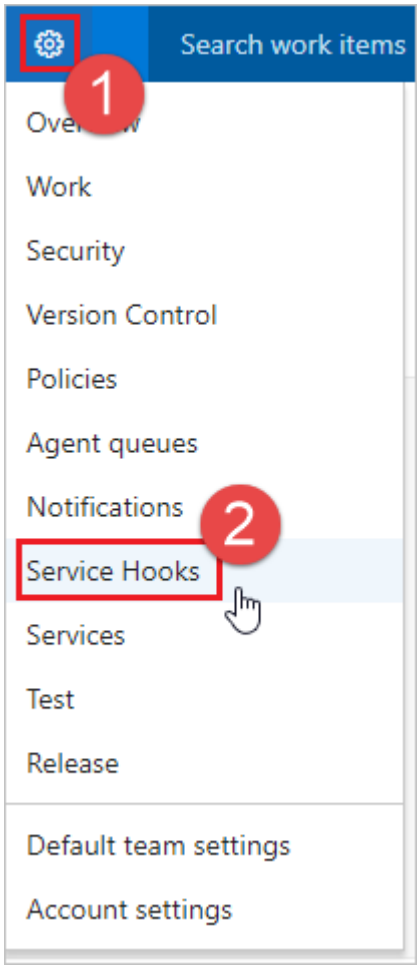
```
ngrok http 3000
```

   Ngrok will create a public URL that forwards to `localhost:3000`. Note that URL as you will need it in the next step. It will look something like this:

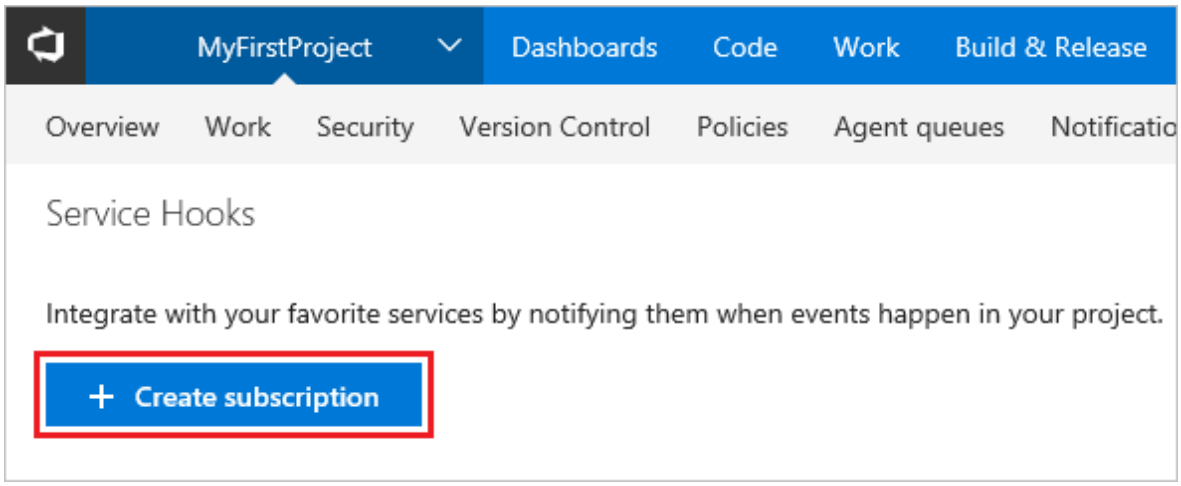
```
http://c3c1bffa.ngrok.io
```

3. Browse to your project in Azure DevOps, e.g. `https://dev.azure.com/<your account>/<your project name>`

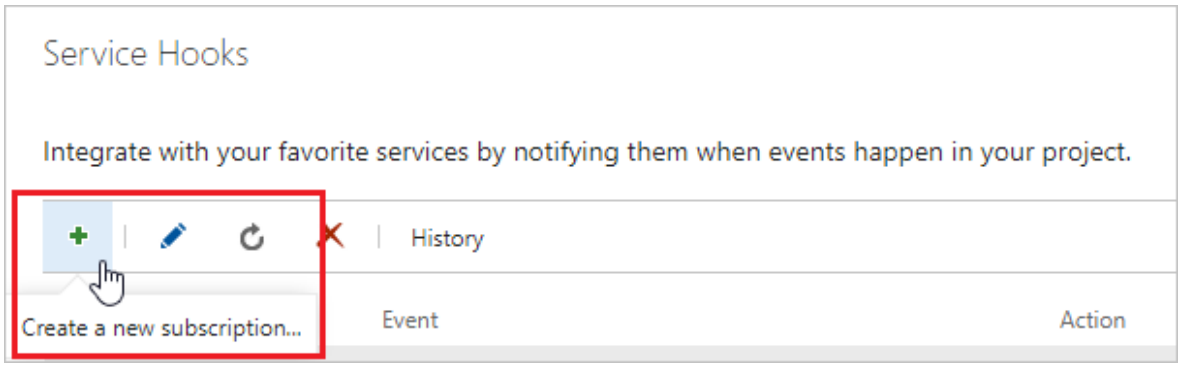
4. From the navigation menu, hover over the **gear** and select **Service Hooks**.



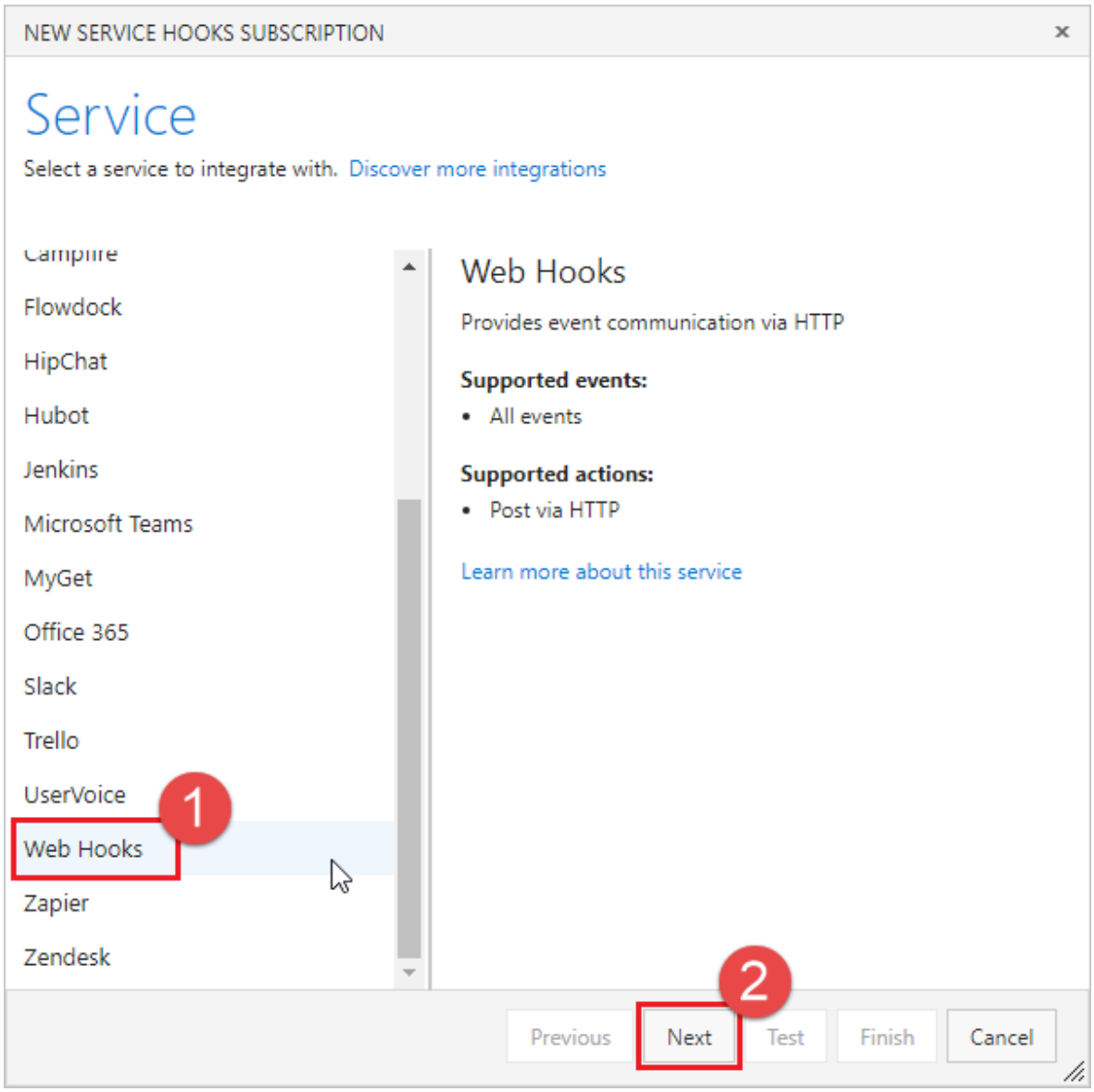
5. If this is your first service hook, select + **Create subscription**.



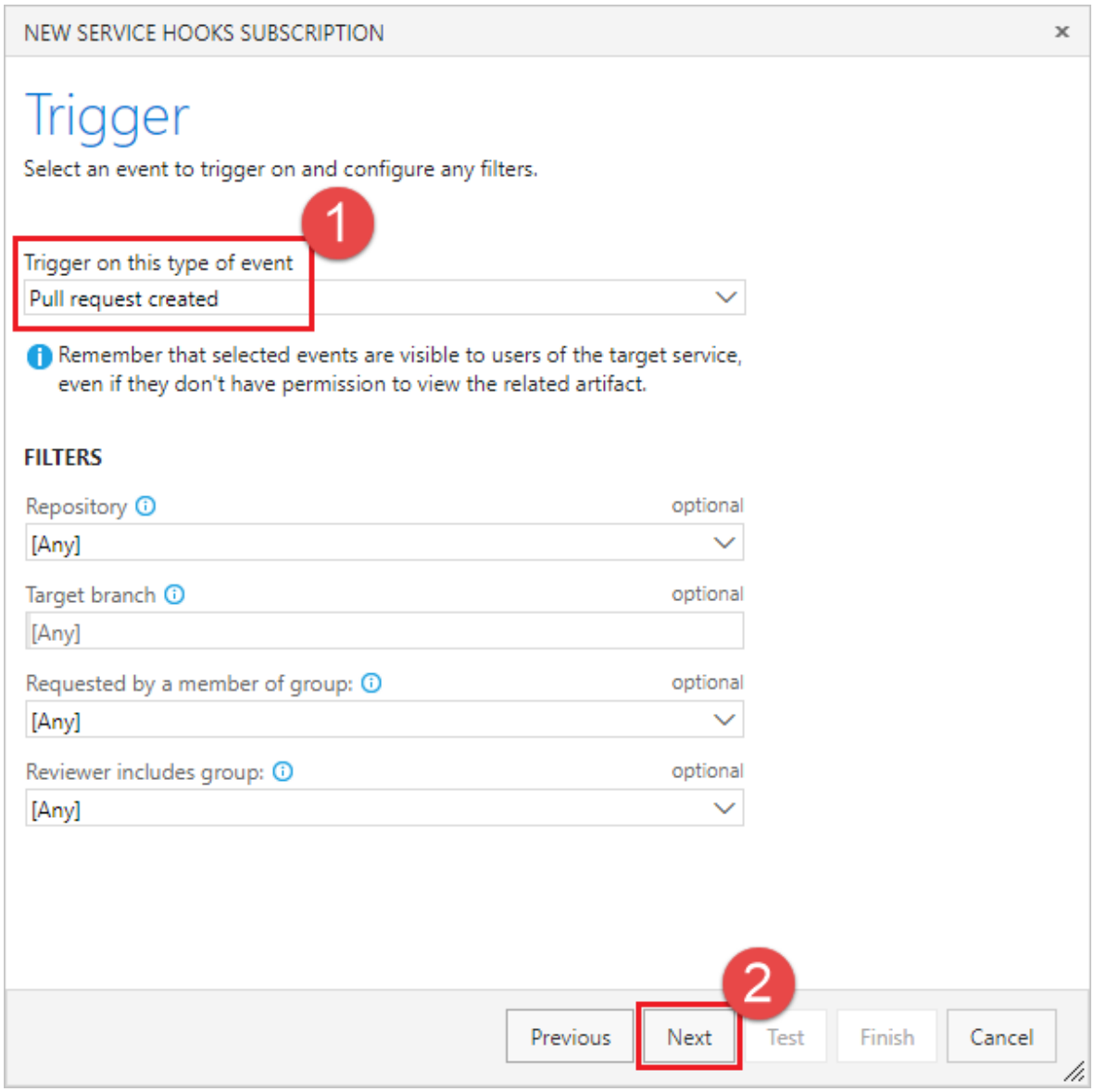
If you already have other service hooks configured, select the green plus `(+)` to create a new service hook subscription.



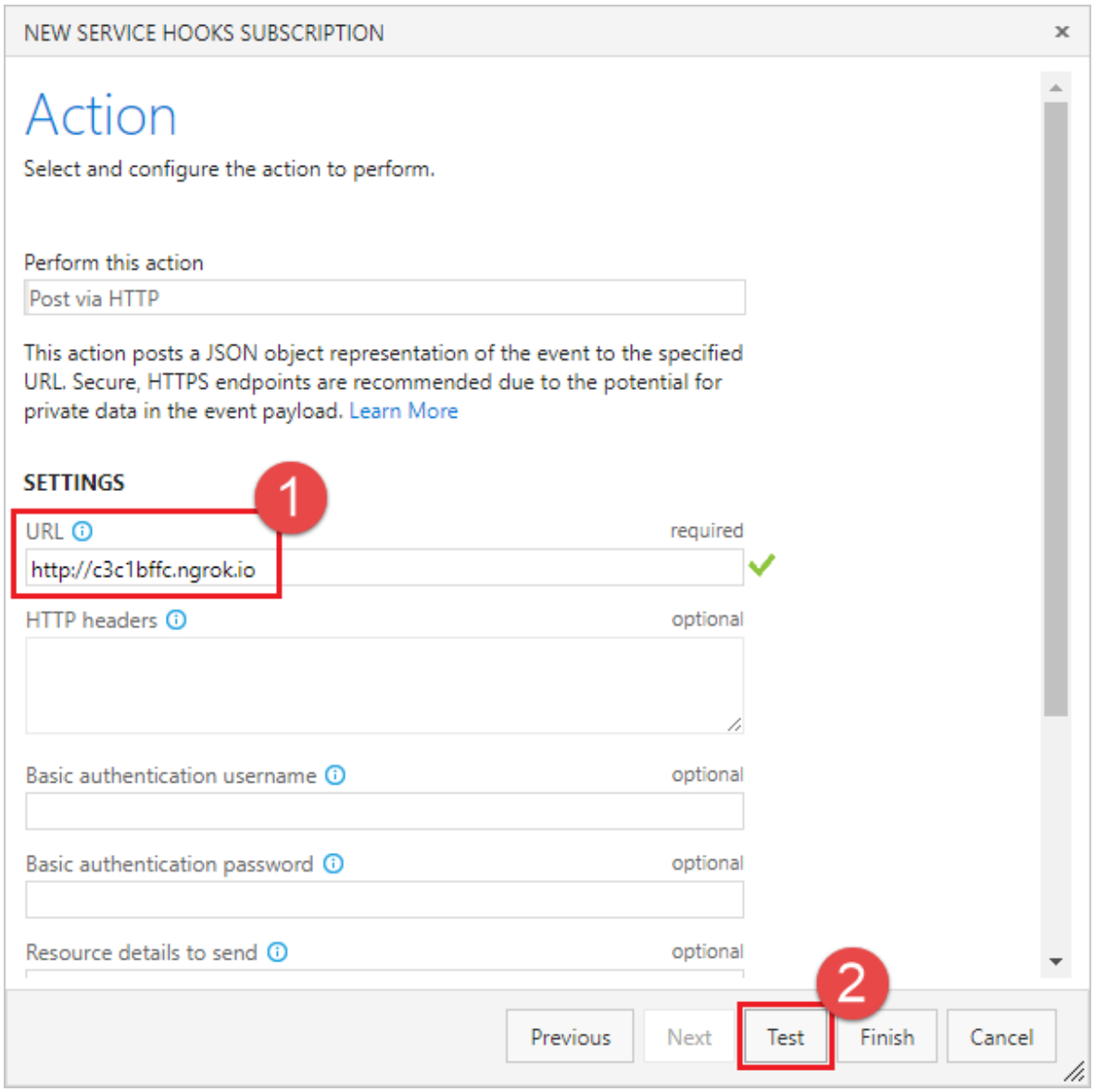
6. On the New Service Hooks Subscription dialog, select **Web Hooks** from the list of services, then select **Next**.



7. Select **Pull request created** from the list of event triggers, then select **Next**.



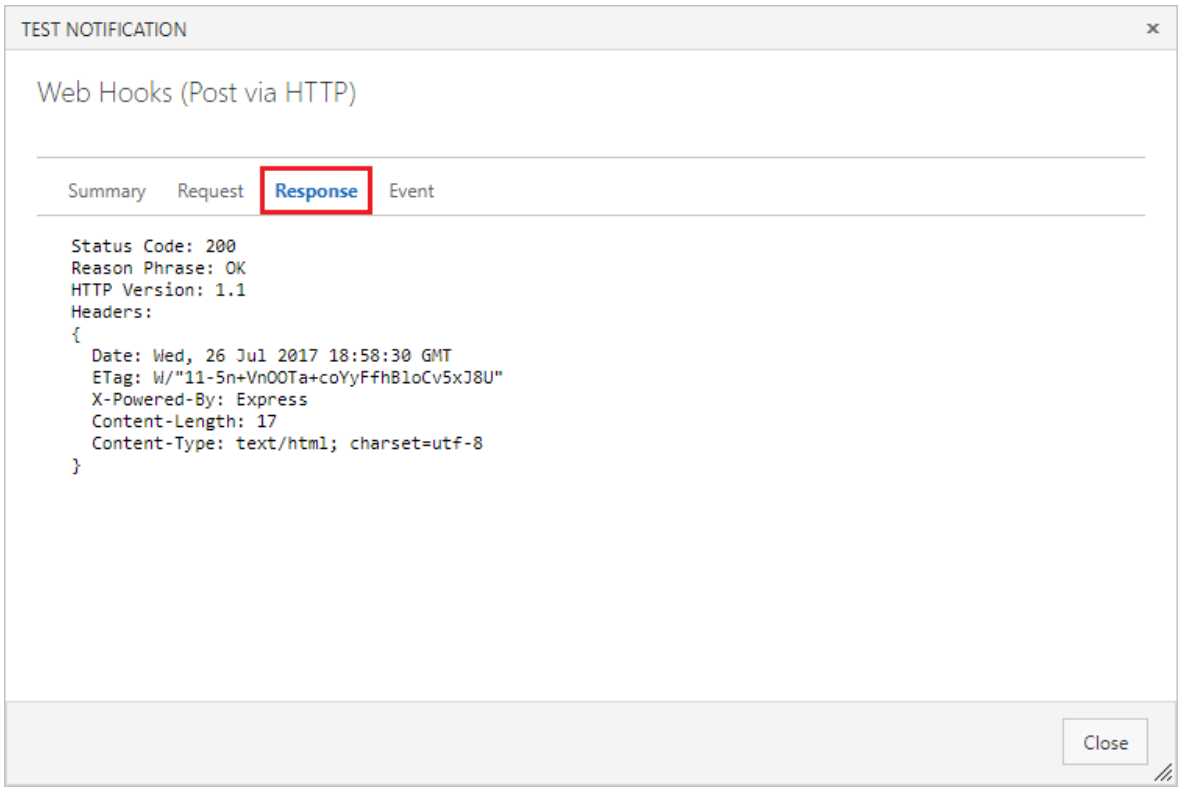
8. In the Action page, enter the URL from ngrok in the **URL** box. Select **Test** to send a test event to your server.



In the ngrok console window, you'll see an incoming `POST` that returned a `200 OK`, indicating your server received the service hook event.

```
HTTP Requests
-------------

POST /                          200 OK
```

In the Test Notification window, select the Response tab to see the details of the response from your server. You should see a content length of 17 that matches the length of the string from your POST handler (i.e. "Received the POST").



9. Close the Test Notification window, and select **Finish** to create the service hook.

Go through steps 3-9 again but this time configure the **Pull request updated** event.

> **ⓘ Important**
>
> Be sure to go through the preceding steps twice and create service hooks for both the **Pull request created** and **Pull request updated** events.

Post status to PRs

Now that your server can receive service hook events when new PRs are created, update it to post back status to the PR.

1. Service hook requests include a JSON payload describing the event. To help parse the JSON returned by the service hook, install the body-parser package.

```
npm install body-parser
```

2. Update `app.js` to use body-parser for parsing `application/json`.

```
JavaScript
```

```
var bodyParser = require('body-parser')

app.use(bodyParser.json())
```

3. To simplify making REST API calls to Azure Repos, install the azure-devops-node-api package.

```
npm install azure-devops-node-api
```

4. Update `app.js` to use the azure-devops-node-api package, set up the details for a connection to your account, and get an instance of the Git API.

JavaScript

```
const vsts = require("azure-devops-node-api")

const collectionURL = process.env.COLLECTIONURL
const token = process.env.TOKEN

var authHandler = vsts.getPersonalAccessTokenHandler(token)
var connection = new vsts.WebApi(collectionURL, authHandler)

var vstsGit = connection.getGitApi().then(
    vstsGit => {
        vstsGit.createPullRequestStatus(prStatus, repoId,
pullRequestId).then( result => {
            console.log(result);
        },
        error => {
            console.log(error);
        })
    },
    error => {
        console.log(error);
    }
);
```

5. Create an environment variable for your collection URL, replacing `<your account>` with the name of your Azure DevOps organization.

```
setx COLLECTIONURL "https://dev.azure.com/<your account>"
```

6. Create a personal auth token (PAT) for your app to use, following these instructions: Authenticating with personal access tokens. You should create a new PAT for every service that you use to access your account, naming it appropriately.

7. Create an environment variable for your PAT.

```
setx TOKEN "yourtokengoeshere"
```

8. Update the `post()` function to read the PR details from the service hook payload. You'll need these values in order to post back status.

JavaScript

```
var repoId = req.body.resource.repository.id
var pullRequestId = req.body.resource.pullRequestId
var title = req.body.resource.title
```

9. Build the status object to post on the PR.

   `State` is an enum of type GitStatusState. Use `succeeded` to indicate that the PR has passed the status check and is ready to merge.

   The `description` is a string value that will be displayed to the user in the Status section and activity feed in the PR details view.

   The `targetUrl` is a URL that will be used to create a link for the description text in the Status section and activity feed. This is the place where users can go to get more information about the status, for example, a build report or test run. If no URL is specified, the description will appear as text with no link.

   The context `name` and `genre` are used to categorize the status and distinguish it from other services posting status.

JavaScript

```
    var prStatus = {
        "state": "succeeded",
        "description": "Ready for review",
        "targetUrl": "https://visualstudio.microsoft.com",
        "context": {
            "name": "wip-checker",
            "genre": "continuous-integration"
        }
    }
```

10. Instead of just blindly posting the `succeeded` status, inspect the PR title to see if the user has indicated if the PR is a work in progress by adding **WIP** to the title. If so, change the status posted back to the PR.

```
JavaScript

    if (title.includes("WIP")) {
        prStatus.state = "pending"
        prStatus.description = "Work in progress"
    }
```

11. Finally, post the status using the `createPullRequestStatus()` method. It requires the status object, the repo ID, and the pull request ID. Output the response to the node console so you can see the result of the post.

```
JavaScript

vstsGit.createPullRequestStatus(prStatus, repoId, pullRequestId).then(
result => {
    console.log(result)
})
```

12. The resulting method should look something like this:

```
JavaScript

app.post("/", function (req, res) {

    // Get the details about the PR from the service hook payload
    var repoId = req.body.resource.repository.id
    var pullRequestId = req.body.resource.pullRequestId
    var title = req.body.resource.title

    // Build the status object that we want to post.
    // Assume that the PR is ready for review...
    var prStatus = {
        "state": "succeeded",
        "description": "Ready for review",
        "targetUrl": "https://visualstudio.microsoft.com",
        "context": {
            "name": "wip-checker",
            "genre": "continuous-integration"
        }
    }

    // Check the title to see if there is "WIP" in the title.
    if (title.includes("WIP")) {
```

```
        // If so, change the status to pending and change the
description.
        prStatus.state = "pending"
        prStatus.description = "Work in progress"
    }

    // Post the status to the PR
    vstsGit.createPullRequestStatus(prStatus, repoId,
pullRequestId).then( result => {
        console.log(result)
    })

    res.send("Received the POST")
})
```

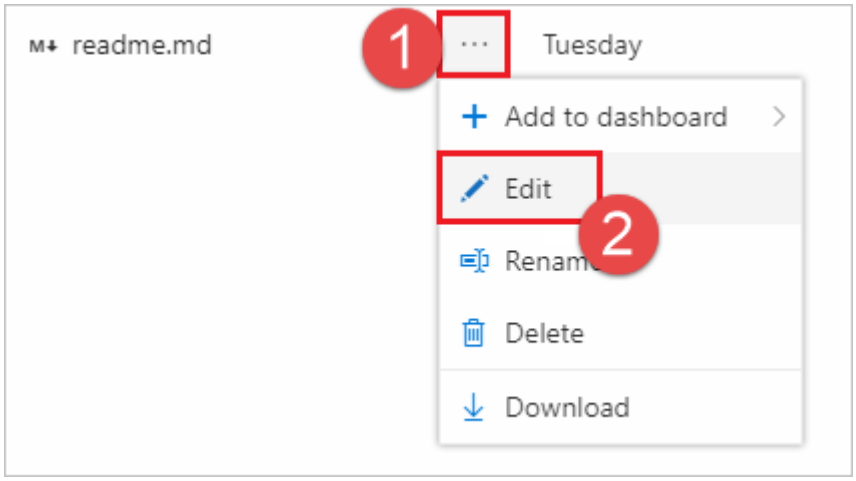
13. Save `app.js` and restart your node app.

```
node app.js
```

Create a new PR to test the status server

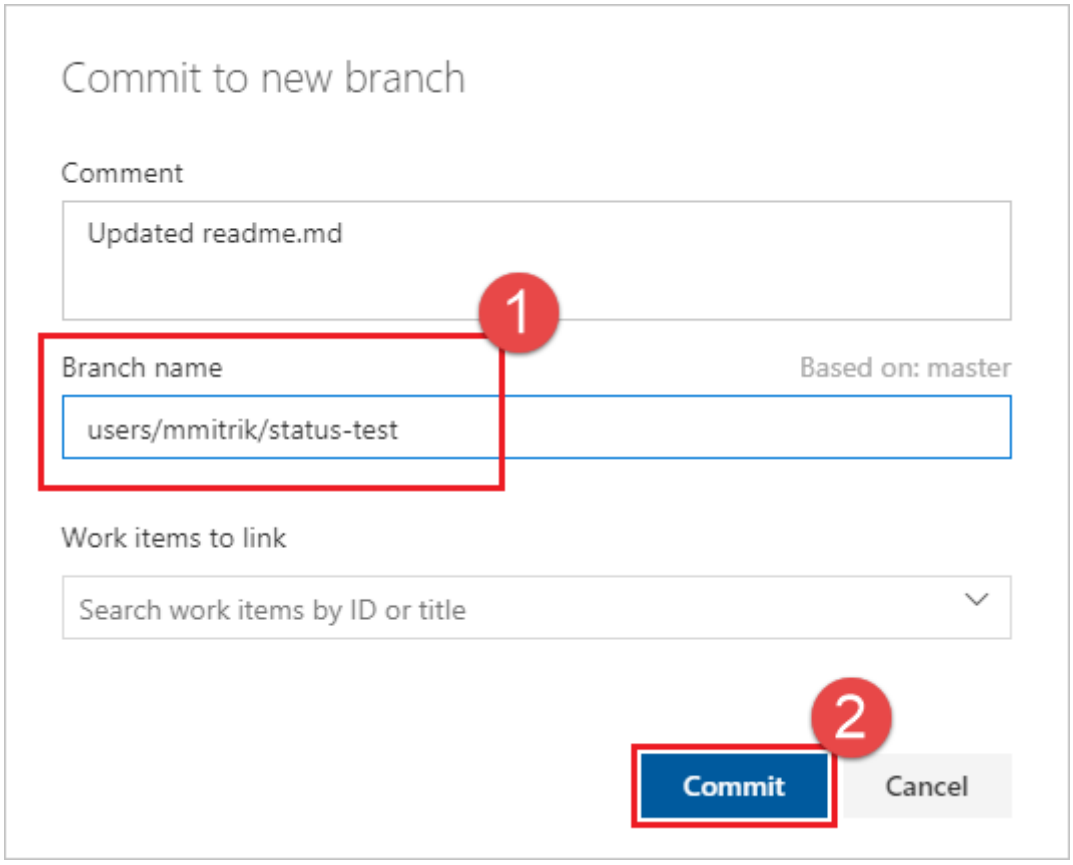
Now that your server is running and listening for service hook notifications, create a pull request to test it out.

1. Start in the files view. Edit the readme.md file in your repo (or any other file if you don't have a readme.md).

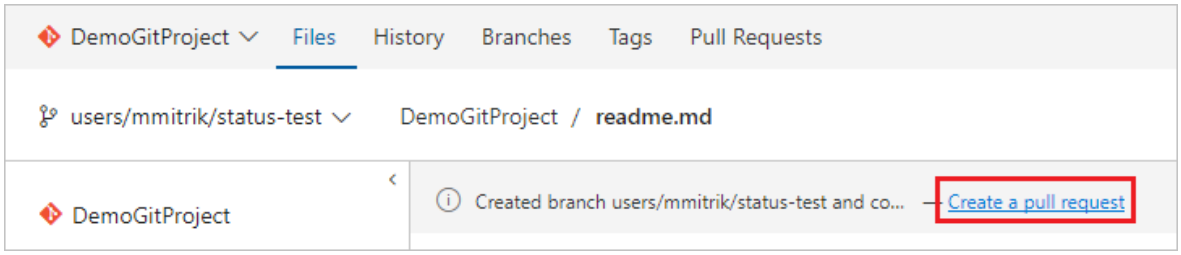


2. Make an edit and commit the changes to the repo.

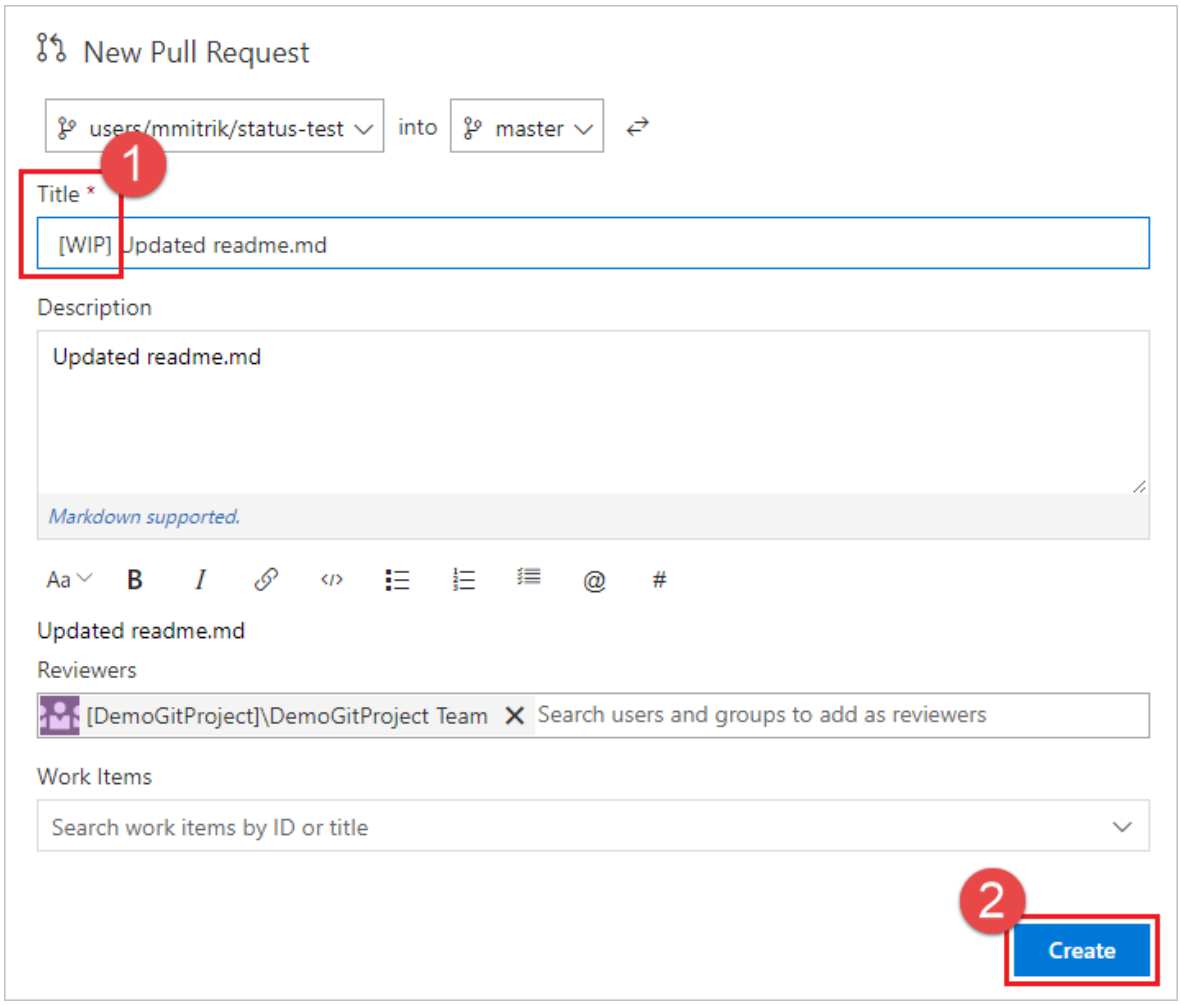
3. Be sure to commit the changes to a new branch so you can create a PR in the next step.



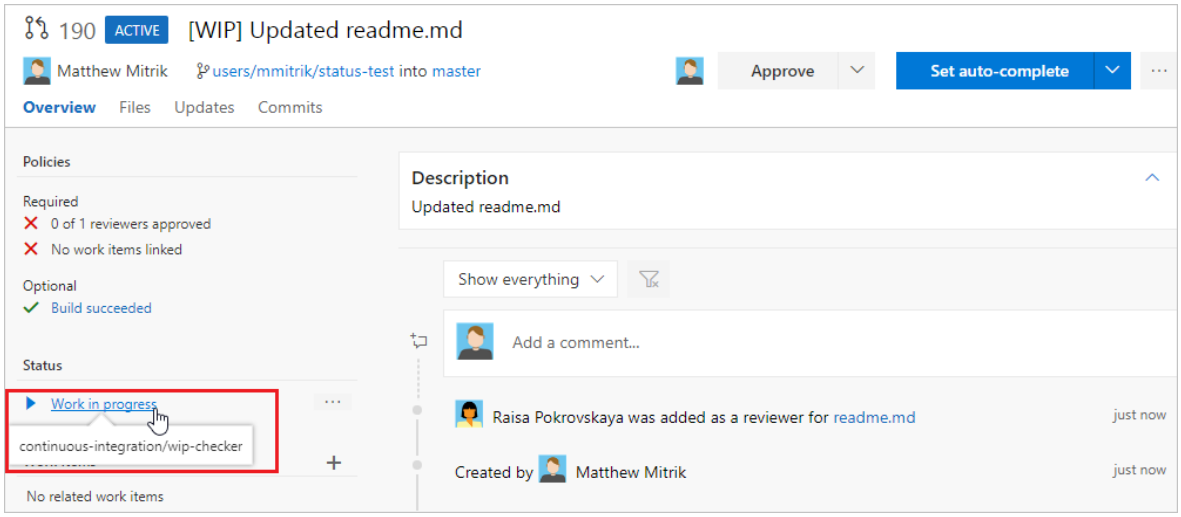
4. Select the **Create a pull request** link.



5. Add **WIP** in the title to test the functionality of the app. Select **Create** to create the PR.



6. Once the PR has been created, you will see the status section, with the **Work in progress** entry which links to the URL specified in the payload.



7. Update the PR title and remove the **WIP** text and note that the status changes from **Work in progress** to **Ready for review**.

Next Steps

- In this article, you learned the basics of how to create a service that listens for PR events via service hooks and can post status messages using the status API. For more information about the pull request status API see the REST API documentation.
- Configure a branch policy for an external service.

Use Azure Functions to create custom branch policies

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The pull request (PR) workflow provides developers with an opportunity to get feedback on their code from peers as well as from automated tools. 3rd party tools and services can participate in the PR workflow by using the PR Status API. This article guides you through the process of creating a custom branch policy using Azure Functions to validate PRs in an Azure DevOps Services Git repository. With Azure Functions you don't have to worry about provisioning and maintaining servers, especially when your workload grows. Azure Functions provide a fully managed compute platform with high reliability and security.

For more information about PR status, see Customize and extend pull request workflows with pull request status.

Prerequisites

An organization in Azure DevOps with a Git repo. If you don't have an organization, sign up to upload and share code in free unlimited private Git repositories.

Create a basic Azure function to listen to Azure Repos events

Follow the create your first Azure function documentation to create a simple function. Modify the code in the sample to look like this:

C#

```
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using Newtonsoft.Json;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req,
TraceWriter log)
```

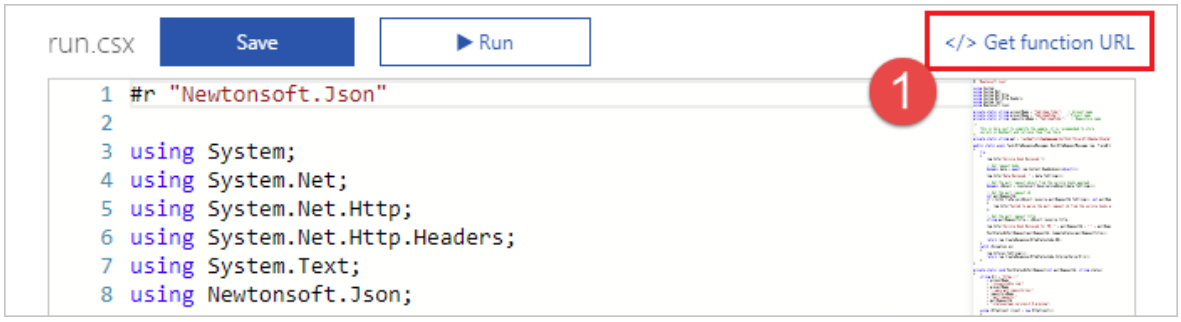
```
{
    try
    {
        log.Info("Service Hook Received.");

        // Get request body
        dynamic data = await req.Content.ReadAsAsync<object>();

        log.Info("Data Received: " + data.ToString());

        // Get the pull request object from the service hooks payload
        dynamic jObject = JsonConvert.DeserializeObject(data.ToString());

        // Get the pull request id
        int pullRequestId;
        if (!Int32.TryParse(jObject.resource.pullRequestId.ToString(), out
pullRequestId))
        {
            log.Info("Failed to parse the pull request id from the service
hooks payload.");
        };

        // Get the pull request title
        string pullRequestTitle = jObject.resource.title;

        log.Info("Service Hook Received for PR: " + pullRequestId + " " +
pullRequestTitle);

        return req.CreateResponse(HttpStatusCode.OK);
    }
    catch (Exception ex)
    {
        log.Info(ex.ToString());
        return req.CreateResponse(HttpStatusCode.InternalServerError);
    }
}
```

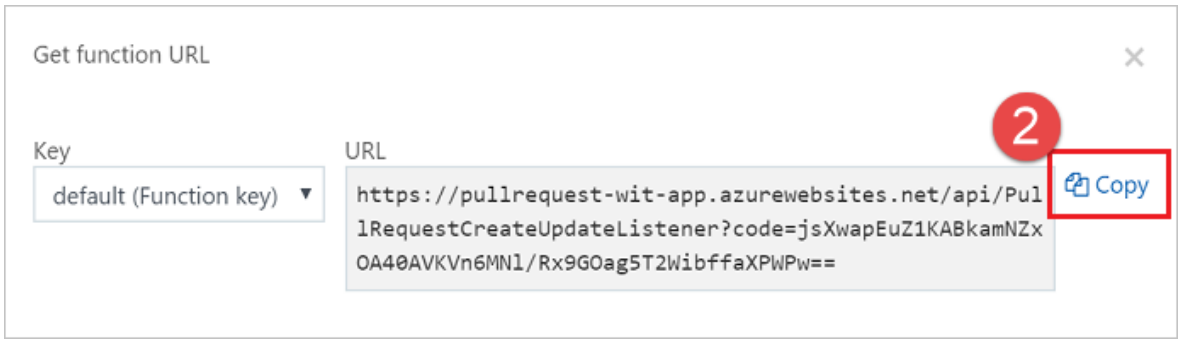
Configure a service hook for PR events

Service hooks are an Azure DevOps Services feature that can alert external services when certain events occur. For this sample, you'll want to set up a service hook for PR events, your Azure function will be notified when a pull request changes. In order to receive `POST` requests when pull requests change, you will need to provide the service hook with the Azure function URL.

For this sample you will need to configure 2 service hooks. The first will be for the **Pull request created** event and the second will be for the **Pull request updated** event.

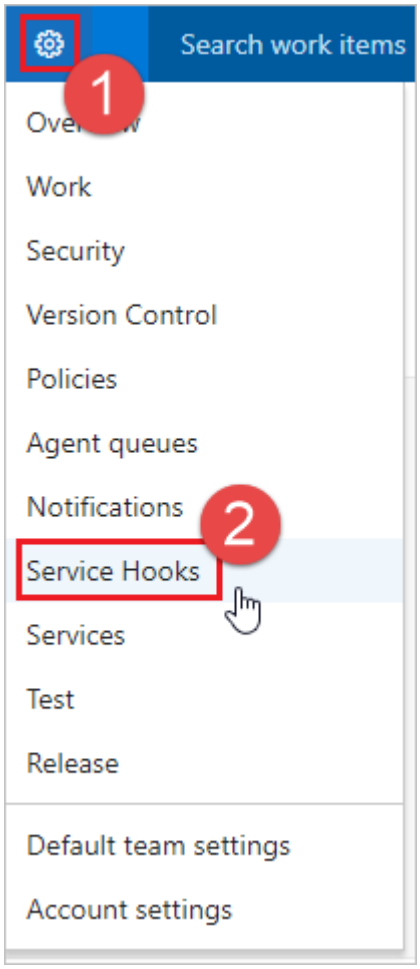
1. Get the function URL from the Azure portal by clicking the **Get function URL** in your Azure function view and copy the URL.



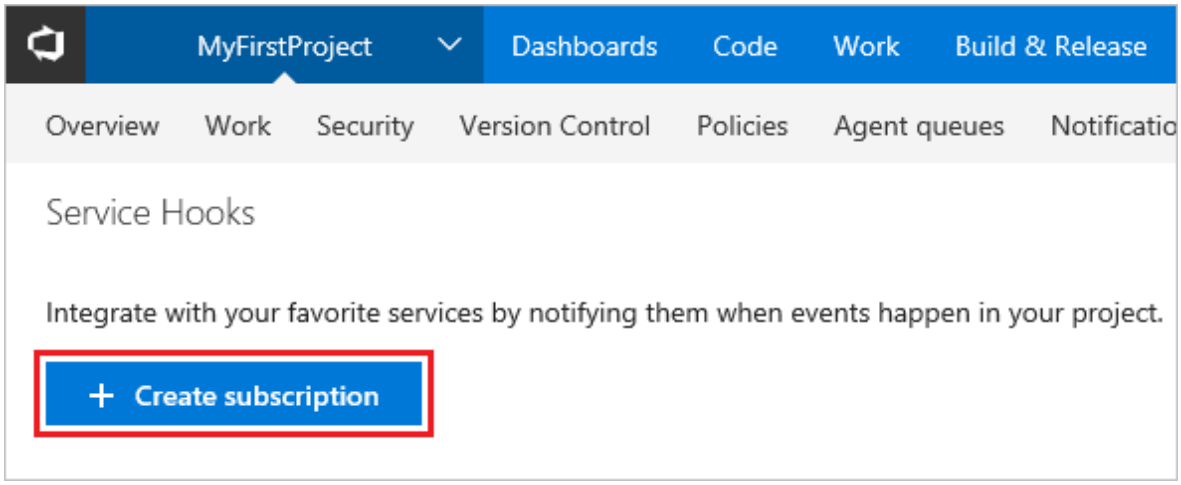


2. Browse to your project in Azure DevOps, e.g. `https://dev.azure.com/<your organization>/<your project name>`

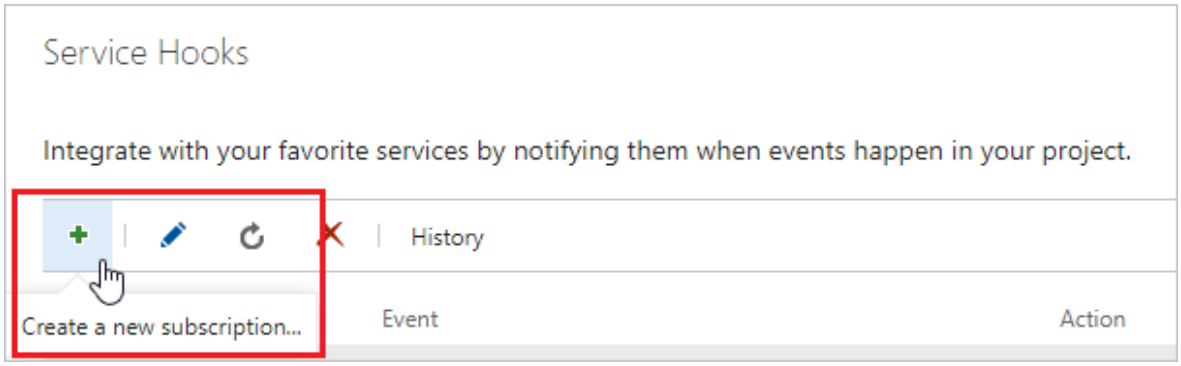
3. From the navigation menu, hover over the **gear** and select **Service Hooks**.



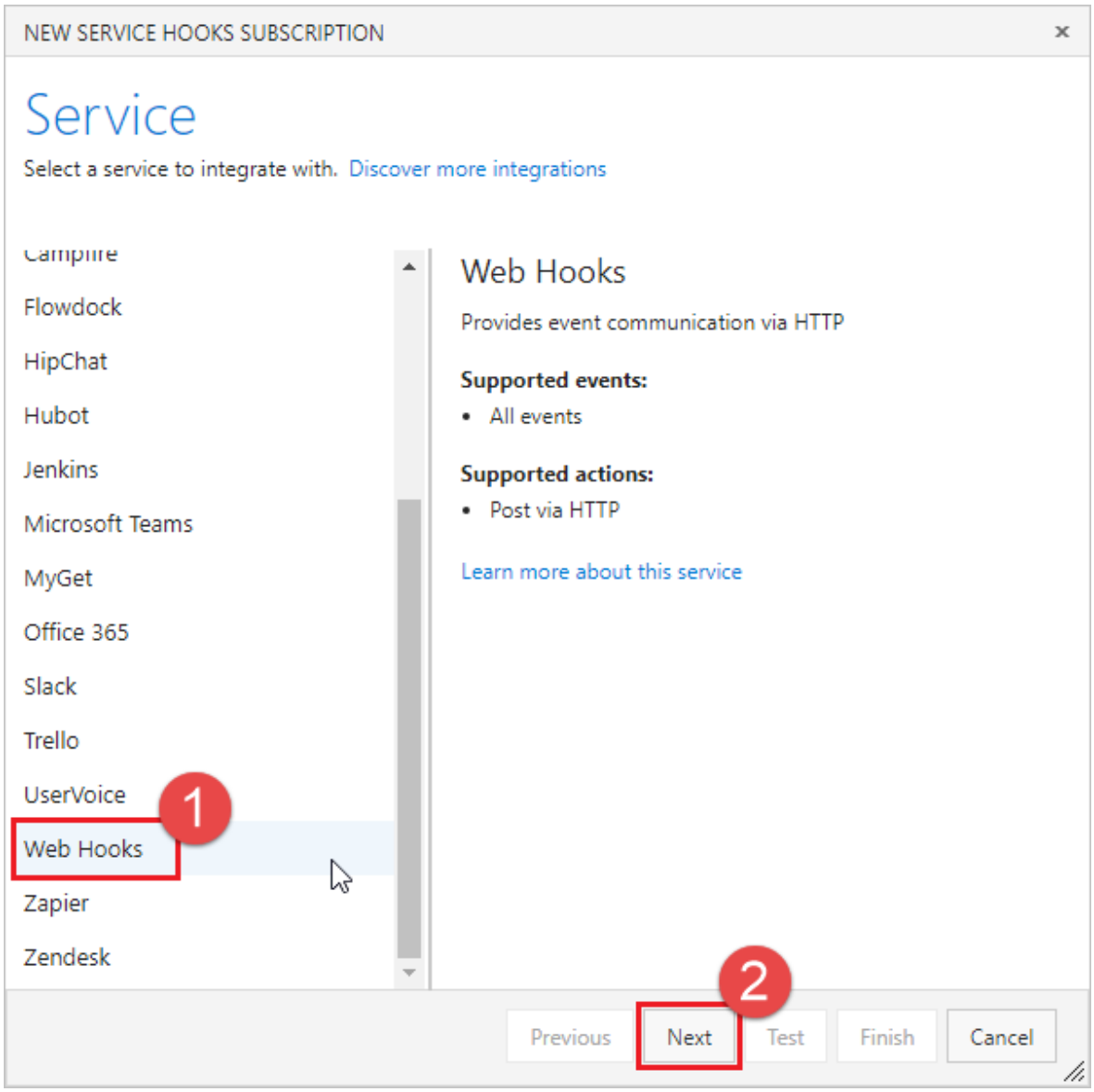
4. If this is your first service hook, select + **Create subscription**.



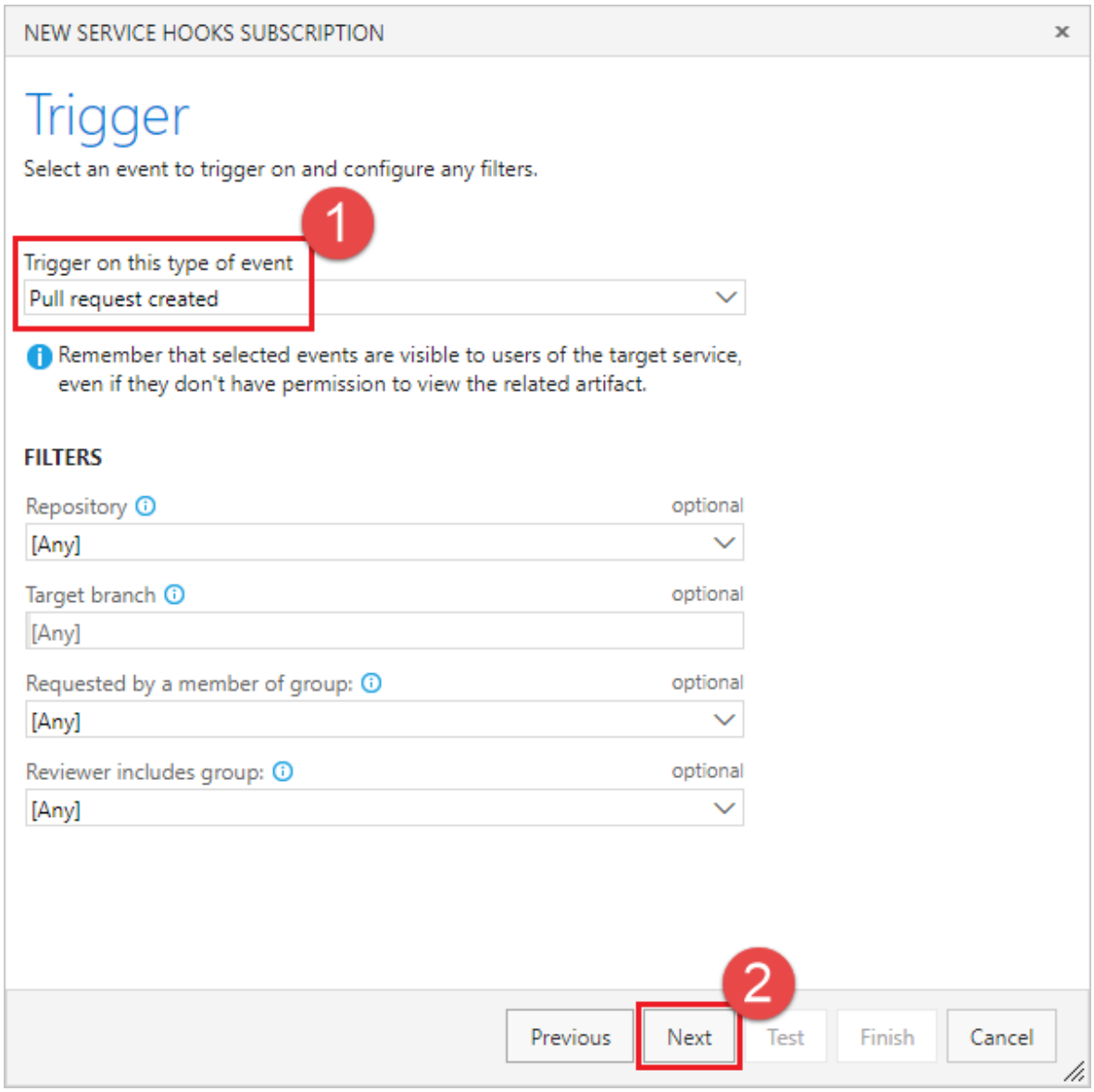
If you already have other service hooks configured, select the green plus `(+)` to create a new service hook subscription.



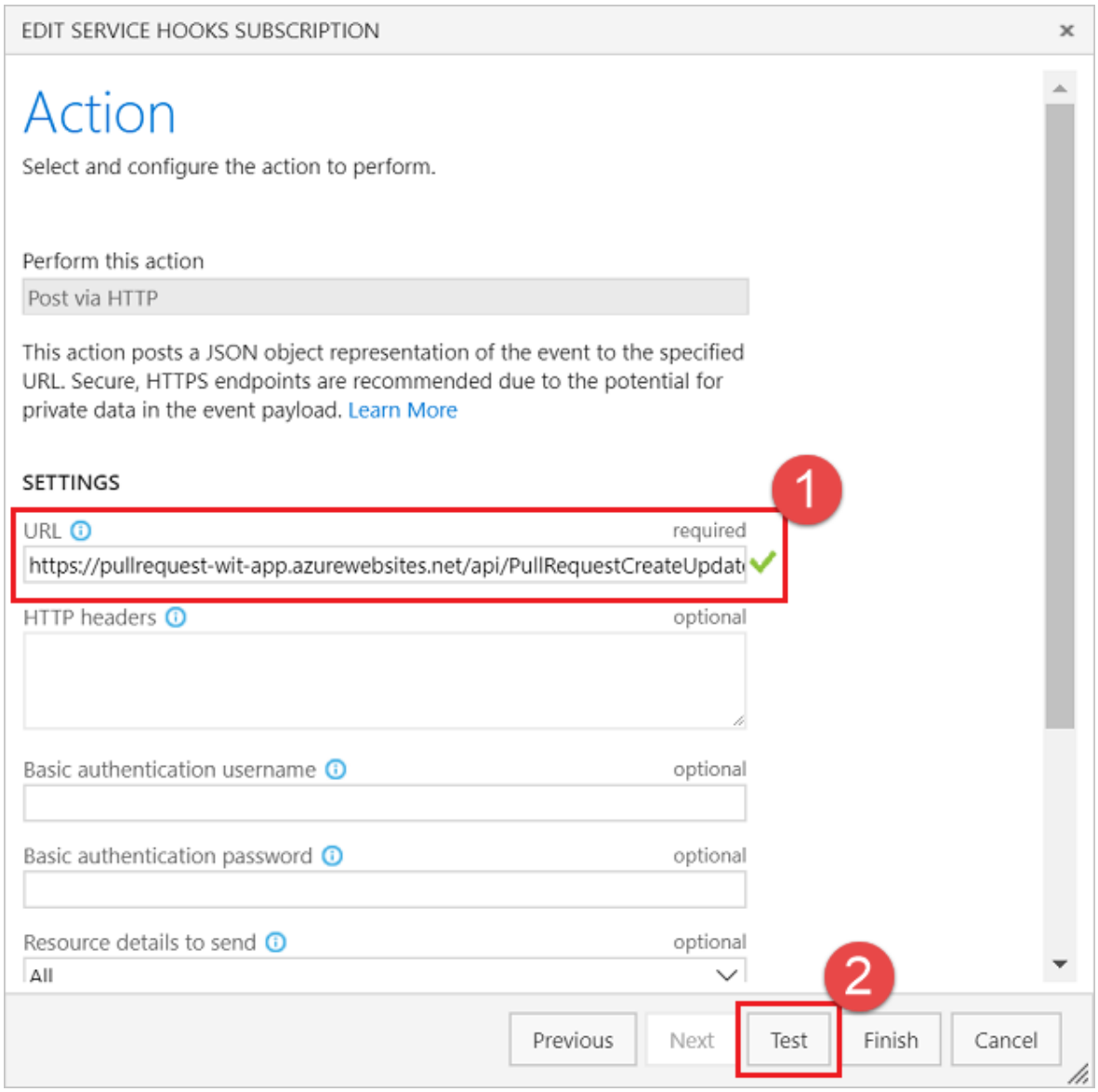
5. On the New Service Hooks Subscription dialog, select **Web Hooks** from the list of services, then select **Next**.



6. Select **Pull request created** from the list of event triggers, then select **Next**.



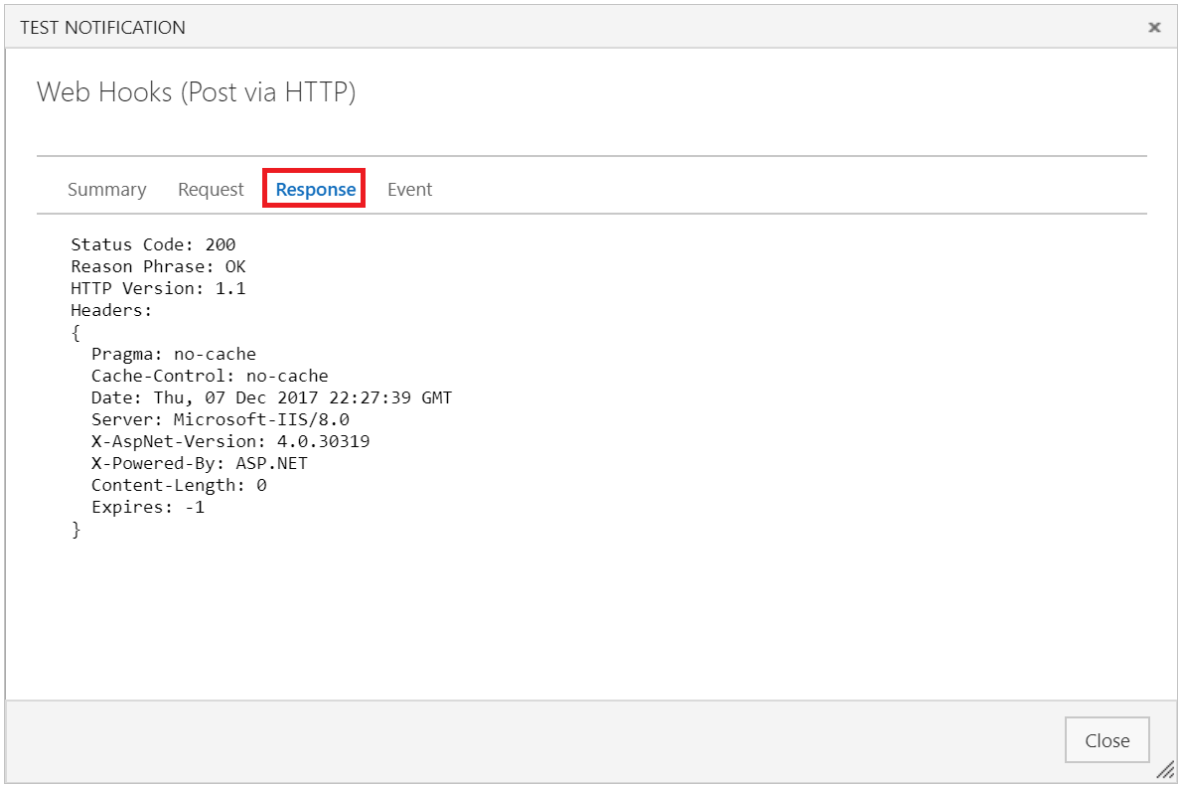
7. In the Action page, enter the URL that you copied in step 1 in the **URL** box. Select **Test** to send a test event to your server.



In the Azure function log window, you'll see an incoming `POST` that returned a `200 OK`, indicating your function received the service hook event.

```
HTTP Requests
-------------

POST /                          200 OK
```

In the Test Notification window, select the Response tab to see the details of the response from your server. You should see the response from your server.



8. Close the Test Notification window, and select **Finish** to create the service hook.

Go through steps 2-8 again but this time configure the **Pull request updated** event.

> **ⓘ Important**
>
> Be sure to go through the preceding steps twice and create service hooks for both the **Pull request created** and **Pull request updated** events.

Create a pull request to verify your Azure function is receiving notifications.

Post status to PRs

Now that your server can receive service hook events when new PRs are created, update it to post back status to the PR. You can use the JSON payload posted by the service hook in order to determine what status to set on your PR.

Update the code of your Azure function to look like the following example.

Make sure to update the code with your organization name, project name, repository name and PAT token. In order to have permission to change PR status, the PAT requires vso.code_status scope, which you can grant by selecting the **Code (status)** scope on the **Create a personal access token** page.

> **ⓘ Important**

This sample code stores the PAT in code to simplify the sample. It is recommended to store secrets in KeyVault and retrieve them from there.

This sample inspects the PR title to see if the user has indicated if the PR is a work in progress by adding **WIP** to the title. If so, the sample code changes the status posted back to the PR. Replace the code in your Azure function with the following code to implement updating the status posted back to the PR.

C#

```
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using Newtonsoft.Json;

private static string organizationName = "[Organization Name]";  //
Organization name
private static string projectName      = "[Project Name]";       // Project
name
private static string repositoryName   = "[Repo Name]";          //
Repository name

/*
    This is here just to simplify the sample, it is recommended to store
    secrets in KeyVault and retrieve them from there.
*/
private static string pat = "[PAT TOKEN]";

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req,
TraceWriter log)
{
    try
    {
        log.Info("Service Hook Received.");

        // Get request body
        dynamic data = await req.Content.ReadAsAsync<object>();

        log.Info("Data Received: " + data.ToString());

        // Get the pull request object from the service hooks payload
        dynamic jObject = JsonConvert.DeserializeObject(data.ToString());

        // Get the pull request id
        int pullRequestId;
        if (!Int32.TryParse(jObject.resource.pullRequestId.ToString(), out
pullRequestId))
        {
            log.Info("Failed to parse the pull request id from the service
hooks payload.");
```

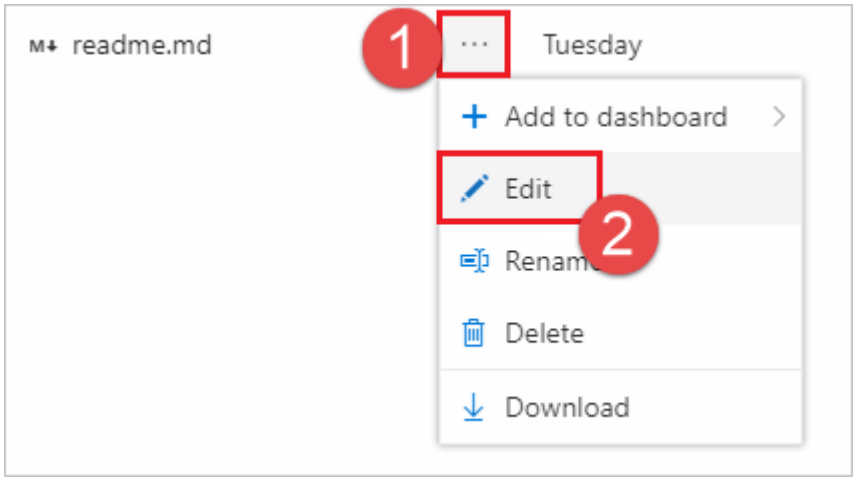
```
        };

        // Get the pull request title
        string pullRequestTitle = jObject.resource.title;

        log.Info("Service Hook Received for PR: " + pullRequestId + " " +
pullRequestTitle);

        PostStatusOnPullRequest(pullRequestId,
ComputeStatus(pullRequestTitle));

        return req.CreateResponse(HttpStatusCode.OK);
    }
    catch (Exception ex)
    {
        log.Info(ex.ToString());
        return req.CreateResponse(HttpStatusCode.InternalServerError);
    }
}

private static void PostStatusOnPullRequest(int pullRequestId, string
status)
{
    string Url = string.Format(

@"https://dev.azure.com/{0}/{1}/_apis/git/repositories/{2}/pullrequests/{3}/
statuses?api-version=4.1",
        organizationName,
        projectName,
        repositoryName,
        pullRequestId);

    using (HttpClient client = new HttpClient())
    {
        client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic", Convert.ToBase64String(
                ASCIIEncoding.ASCII.GetBytes(
                string.Format("{0}:{1}", "", pat))));

        var method = new HttpMethod("POST");
        var request = new HttpRequestMessage(method, Url)
        {
            Content = new StringContent(status, Encoding.UTF8,
"application/json")
        };

        using (HttpResponseMessage response =
client.SendAsync(request).Result)
        {
            response.EnsureSuccessStatusCode();
        }
    }
}
```

```
private static string ComputeStatus(string pullRequestTitle)
{
    string state = "succeeded";
    string description = "Ready for review";

    if (pullRequestTitle.ToLower().Contains("wip"))
    {
        state = "pending";
        description = "Work in progress";
    }

    return JsonConvert.SerializeObject(
        new
        {
            State = state,
            Description = description,
            TargetUrl = "https://visualstudio.microsoft.com",

            Context = new
            {
                Name = "PullRequest-WIT-App",
                Genre = "pr-azure-function-ci"
            }
        });
}
```

Create a new PR to test the status server

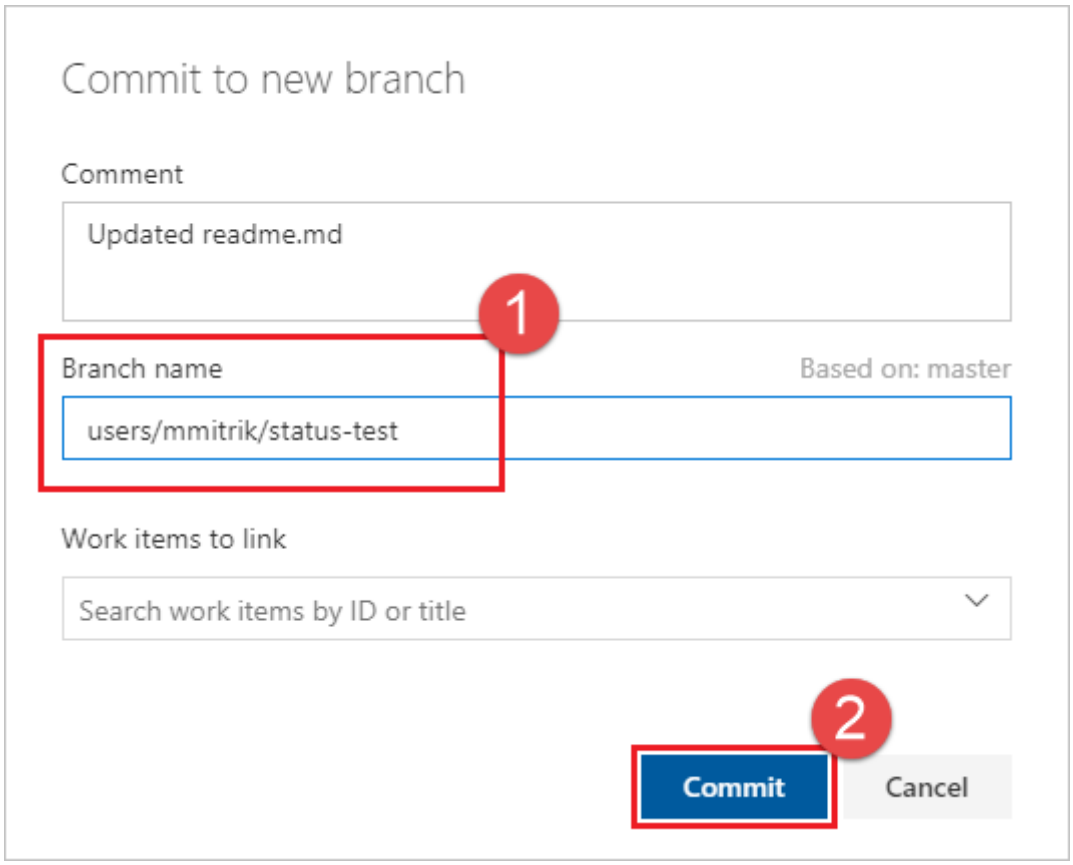
Now that your server is running and listening for service hook notifications, create a pull request to test it out.

1. Start in the files view. Edit the readme.md file in your repo (or any other file if you don't have a readme.md).

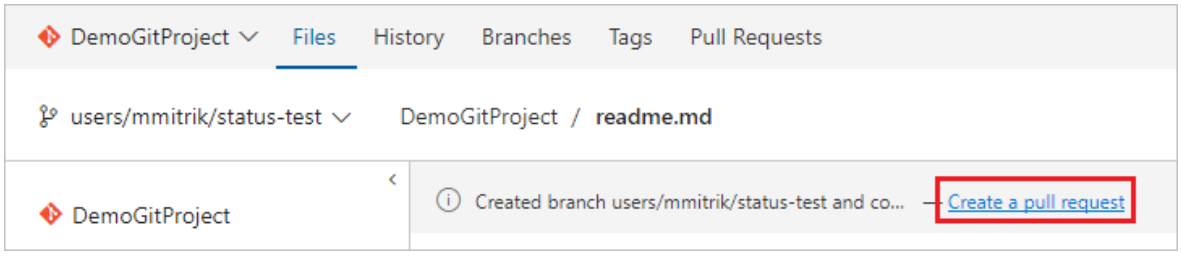


2. Make an edit and commit the changes to the repo.

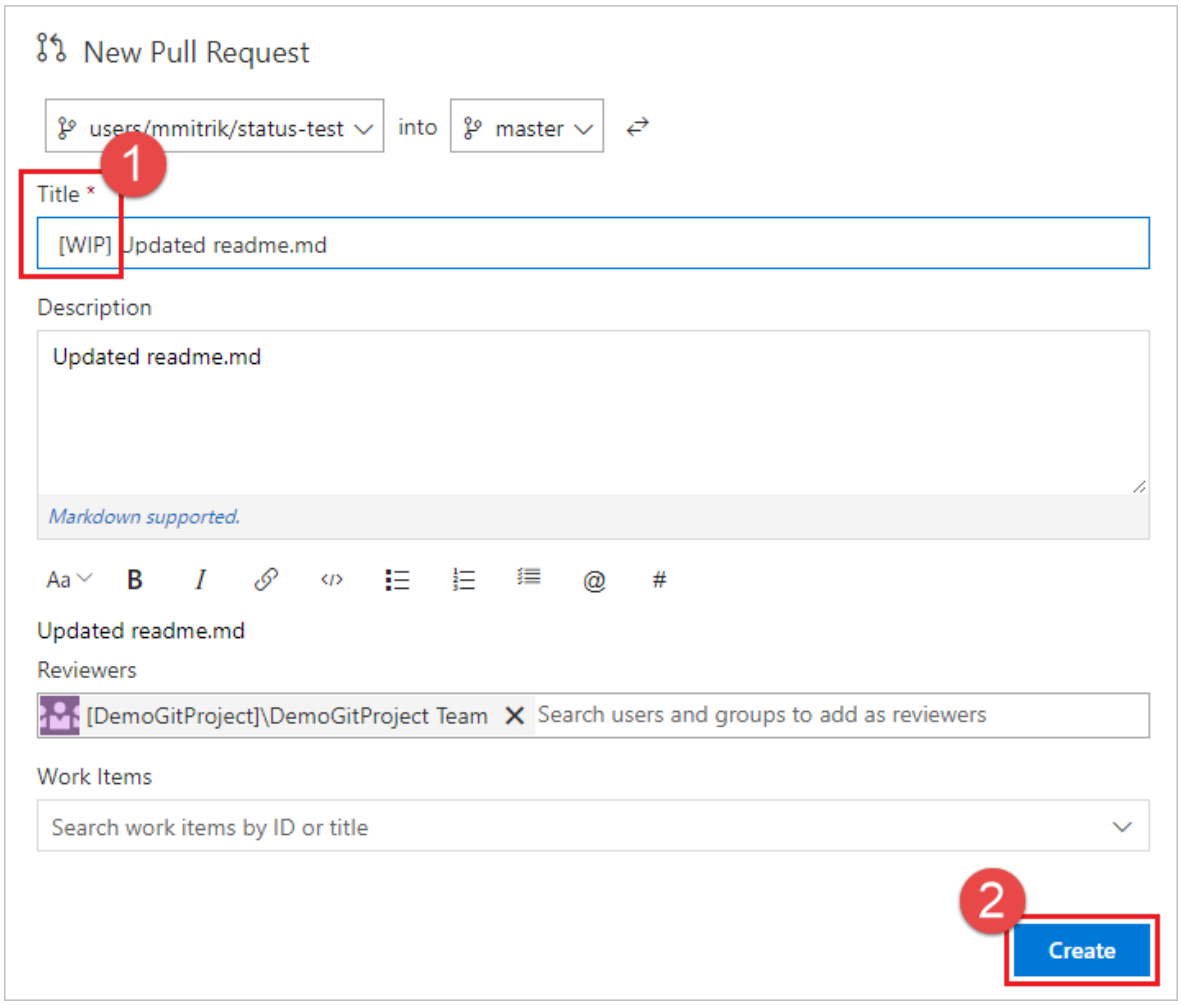
3. Be sure to commit the changes to a new branch so you can create a PR in the next step.



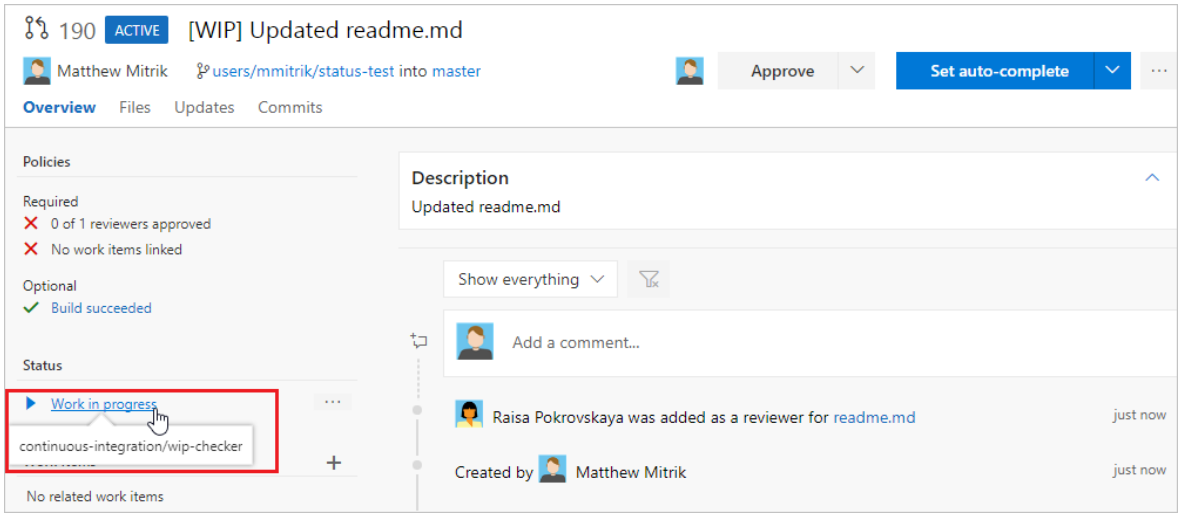
4. Select the **Create a pull request** link.



5. Add **WIP** in the title to test the functionality of the app. Select **Create** to create the PR.



6. Once the PR has been created, you will see the status section, with the **Work in progress** entry which links to the URL specified in the payload.



7. Update the PR title and remove the **WIP** text and note that the status changes from **Work in progress** to **Ready for review**.

Next Steps

- In this article, you learned the basics of how to create a serverless Azure function that listens for PR events via service hooks and can post status messages using the status API. For more information about the pull request status API see the REST API documentation.
- Configure a branch policy for an external service.

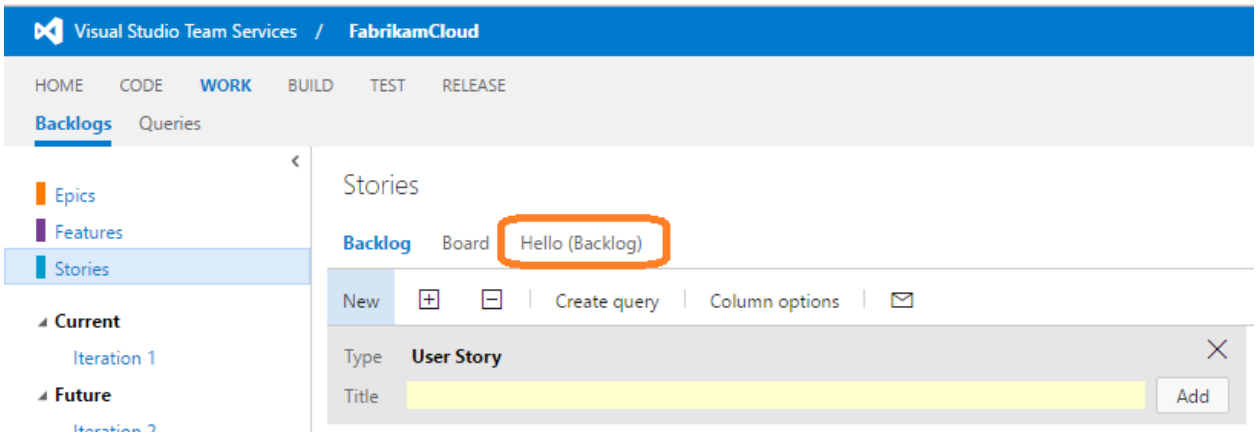
Add tabs on backlog pages

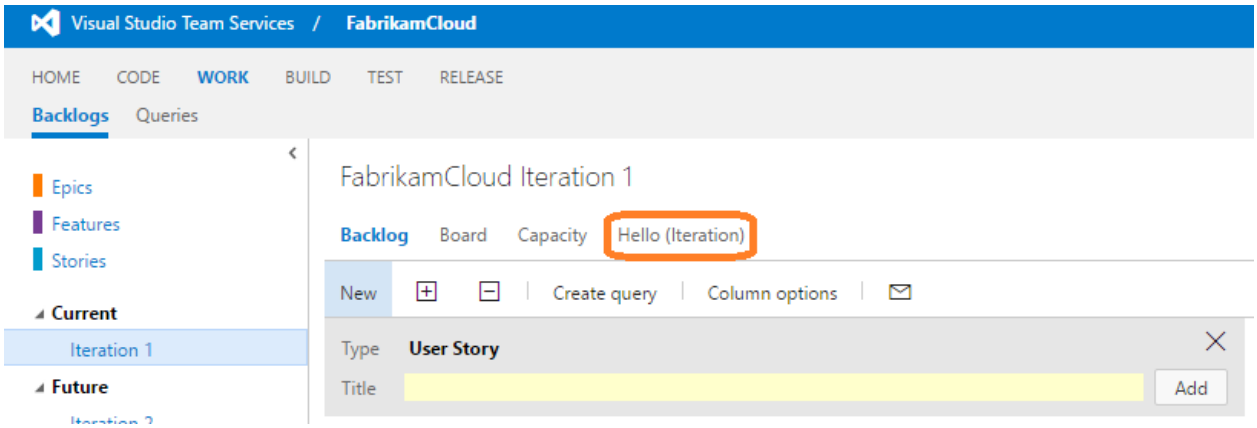
Article • 10/04/2022

Azure DevOps Services

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

If you have a web page that can be hosted in an iframe, it can also be hosted in Azure DevOps Services. This webpage would be a tab on the backlog pages. In this example, we'll add a Hello World tab on the Product Backlog and the Iteration Backlog.





Create your web page

1. Get the Client SDK `SDK.js` file and add it to your web app. Place it in the `home/sdk/scripts` folder.
   a. Use the 'npm install' command to retrieve the SDK: `npm install azure-devops-extension-sdk`.

b. To learn more about the SDK, visit the Client SDK GitHub Page.

2. Add the web page that you want to display as a hub. We're doing a simple `hello-world.html` page here, added to the `home` directory.

3. In your HTML page, add a reference to the SDK, and call `init()` and `notifyLoadSucceeded()`.

HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Hello World</title>
   <script src="sdk/scripts/SDK.js"></script>
</head>
<body>
   <script type="text/javascript">SDK.init();</script>
   <h1>Hello World</h1>
   <script type="text/javascript">SDK.notifyLoadSucceeded();</script>
</body>
</html>
```

Update your extension manifest

Update your extension manifest file with the following code:

JSON

```
...

    "contributions": [
        {
            "id": "Fabrikam.HelloWorld.Backlog.Tabs",
            "type": "ms.vss-web.tab",
            "description": "Adds a 'Hello' tab to the Product and Iteration
backlog tabs.",
            "targets": [
                "ms.vss-work-web.product-backlog-tabs",
                "ms.vss-work-web.iteration-backlog-tabs"
            ],
            "properties": {
                "name": "Hello",
                "uri": "hello-world.html",
                "registeredObjectId": "backlogTabObject"
            }
        }
    ],
    "scopes": [
```

```
            "vso.work"
        ],
        "files": [
            {
                "path": "hello-world.html", "addressable": true
            },
            {
                "path": "scripts", "addressable": true
            },
            {
                "path": "sdk/scripts", "addressable": true
            },
            {
                "path": "images/logo.png", "addressable": true
            }
        ]
...
```

Contributions

The **contributions** stanza contains information about your tasks.

For each contribution in your extension, the manifest defines

- the type of contribution (tab in this case),
- the contribution target (the product or iteration backlog tabs in this case),
- and the properties that are specific to each type of contribution. For a tab, we have

Property	Description
name	Name of the hub
uri	Path (relative to the extension's base URI) of the page to surface as the tab
registeredObjectId	ID of the object registered for the tab. Include code like the example below in the html file indicated in the "uri" property of the contribution shown previously.

Scopes

It includes the scopes that your extension requires. In this case, we need `vso.work` to access work items.

Files

Include all of the files your extension accesses.
For your files, set `addressable` to `true` unless you include other files that don't need to be URL-addressable.

Example registeredObjectId

JavaScript

```
SDK.register("backlogTabObject", {
    pageTitle: function(state) {
        return "Hello Tab";
    },
    updateContext: function(tabContext) {
    },
    isInvisible: function(state) {
        return false;
    },
    isDisabled: function(state) {
        return false;
    }
});
```

Learn about all of the places where you can add a hub in Extensibility points.

Next steps

Package, Publish, and Install or

Test and Debug

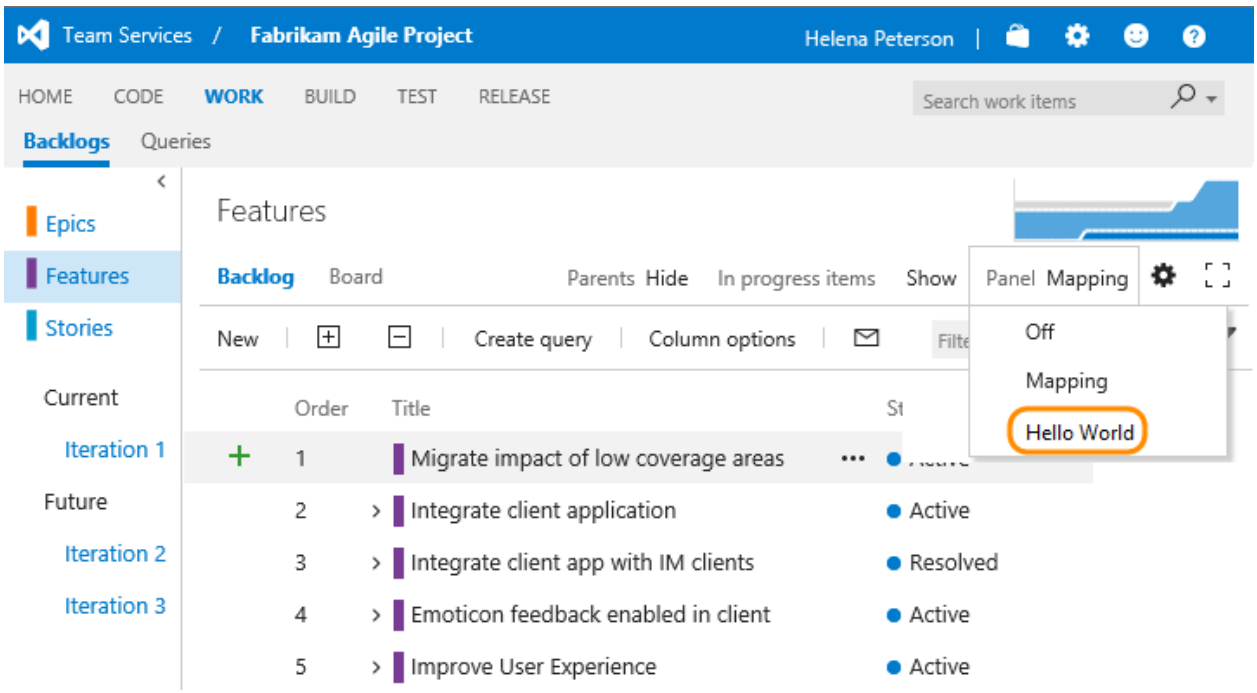
Add panels on backlog pages

Article • 02/20/2023

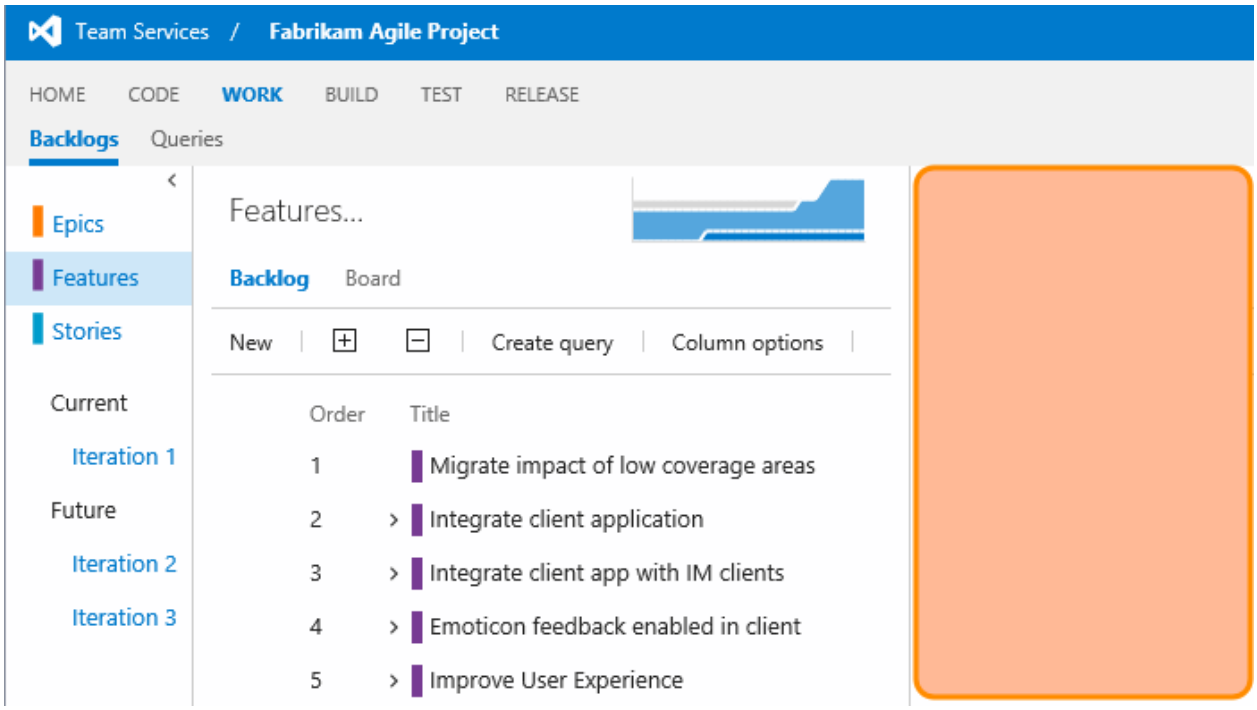
Azure DevOps Services

Here, we add a simple Hello World extension as a panel on the Portfolio backlog, Product backlog, and Iteration backlog.

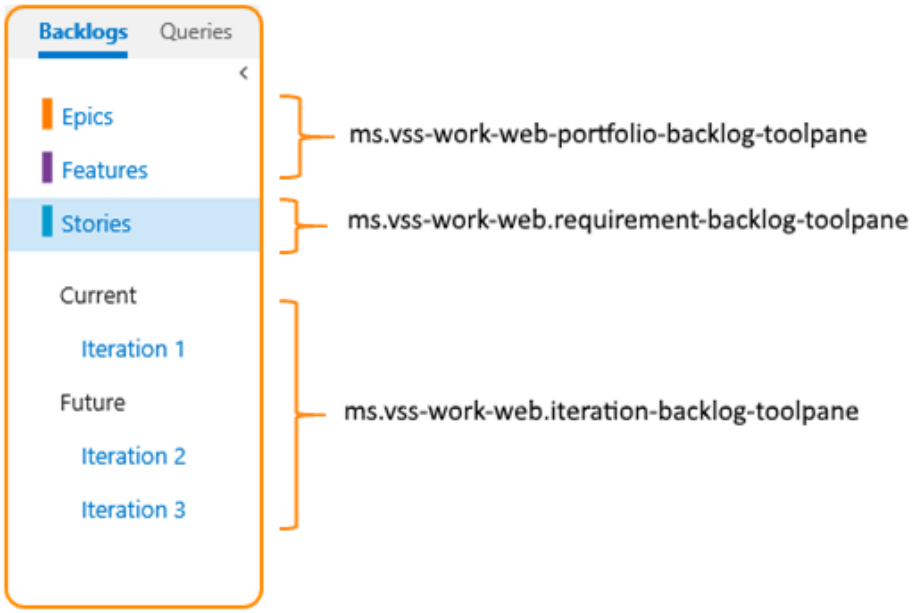
> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.



The custom panel opens in the same space that the mapping panel opens if it were selected.



There are three types of backlogs that can be targets for panel extensions: Portfolio backlogs, Product backlogs, and Iteration backlogs. For the Agile template, this breakdown is as below. This is representative of Scrum and CMMI as well. For custom templates, please consult your process to see which backlogs are requirement or portfolio category.



For more information, see the Azure DevOps Services Extension Sample.

Update your extension manifest

Update your extension manifest file with the following code:

JSON

```
...
    "contributions": [
        {
            "id": "Fabrikam.HelloWorld.Backlogs.Panel",
            "type": "ms.vss-work-web.backlog-panel",
            "description": "Adds a 'Hello' panel to Product and Iteration
backlog pages.",
            "targets": [
                "ms.vss-work-web.requirement-backlog-toolpane",
                "ms.vss-work-web.portfolio-backlog-toolpane",
                "ms.vss-work-web.iteration-backlog-toolpane"
            ],
            "properties": {
                "title": "Hello Panel Pane",
                "name": "Hello Panel",
                "uri": "index.html",
                "registeredObjectId": "backlogPanelObject"
            }
        }
    ],
    "scopes": [
        "vso.work"
    ]
...
```

Contribution

For each contribution in your extension, the manifest defines

- the type of contribution (backlog panel in this case),
- the contribution target (the requirements, portfolio, and iteration backlogs in this case),
- and the properties that are specific to each type of contribution. For panels, we have

Property	Description
title	Tooltip text that appears on the menu item
name	What appears in the dropdown for panel selection
uri	Path (relative to the extension's base URI) of the page to surface as the panel
registeredObjectId	Id of the object registered for the panel

Learn about all of the places where you can add an extension in Extensibility points.

Scopes

Include the scopes that your extension requires. In this case, we need `vso.work` to access work items.

Get selection events

To get selection events (information about what work items are selected) implement this interface on your registered object.

JavaScript

```
...
    IContributedPanel {
        workItemSelectionChanged: (selectedWorkItems) => void;
    }
...
```

Next steps

Package, Publish, and Install or

Test and Debug

Extend the work item form

Article • 05/08/2024

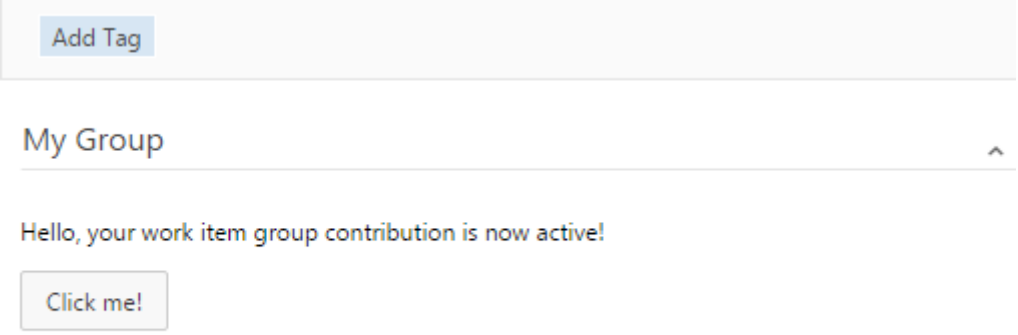
Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Learn how to customize how the work item form gets presented to users via contributions that are made through an extension.

- Add a group to the main page
- Add a page (tab)
- Add an action to the context menu
- Add a custom control
- Listen for events on the form
- Configure contributions in work item form

See the **UI** example in the Azure DevOps Extension Samples on GitHub for the full source.

Add a group



To add a group to the main page, add a contribution to your extension manifest. The type of this contribution should be `ms.vss-work-web.work-item-form-group` and it should target the `ms.vss-work-web.work-item-form` contribution.

JSON

```
"contributions": [
   {
       "id": "sample-work-item-form-group",
       "type": "ms.vss-work-web.work-item-form-group",
       "description": "Custom work item form group",
       "targets": [
           "ms.vss-work-web.work-item-form"
       ],
```

```
        "properties": {
            "name": "My Group",
            "uri": "workItemGroup.html",
            "height": 600
        }
    }
]
```

Properties

Expand table

Property	Description
`name`	Text that appears on the group.
`uri`	URI to a page that hosts the html that shows on the work item form and its scripts.
`height`	(Optional) Defines the height of the group. When omitted, it's 100%.

JavaScript sample

This example shows how to register an object that's called when events occur on the work item form that may impact your contributed group. Please see WorkItemFormGroup Sample for more examples.

JavaScript

```
import { IWorkItemFormService, WorkItemTrackingServiceIds } from "azure-devops-extension-api/WorkItemTracking";
import * as SDK from "azure-devops-extension-sdk";

// Get the WorkItemFormService.  This service allows you to get/set fields/links on the 'active' work item (the work item
// that currently is displayed in the UI).
async function getWorkItemFormService()
{
    const workItemFormService = await SDK.getService<IWorkItemFormService>(WorkItemTrackingServiceIds.WorkItemFormService);
    return workItemFormService;
}

// Register a listener for the work item group contribution after initializing the SDK.
SDK.register(SDK.getContributionId(), function () {
    return {
        // Called when the active work item is modified
        onFieldChanged: function(args) {
```

```
            $(".events").append($("<div/>").text("onFieldChanged - " +
JSON.stringify(args)));
        },

        // Called when a new work item is being loaded in the UI
        onLoaded: function (args) {

            getWorkItemFormService().then(function(service) {
                // Get the current values for a few of the common fields
                service.getFieldValues(["System.Id", "System.Title",
"System.State", "System.CreatedDate"]).then(
                    function (value) {
                        $(".events").append($("<div/>").text("onLoaded - " +
JSON.stringify(value)));
                    });
            });
        },

        // Called when the active work item is being unloaded in the UI
        onUnloaded: function (args) {
            $(".events").empty();
            $(".events").append($("<div/>").text("onUnloaded - " +
JSON.stringify(args)));
        },

        // Called after the work item has been saved
        onSaved: function (args) {
            $(".events").append($("<div/>").text("onSaved - " +
JSON.stringify(args)));
        },

        // Called when the work item is reset to its unmodified state (undo)
        onReset: function (args) {
            $(".events").append($("<div/>").text("onReset - " +
JSON.stringify(args)));
        },

        // Called when the work item has been refreshed from the server
        onRefreshed: function (args) {
            $(".events").append($("<div/>").text("onRefreshed - " +
JSON.stringify(args)));
        }
    }
});
```

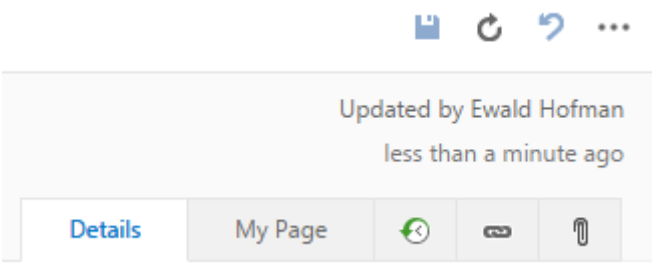
Events

Expand table

Event	Event description	Argument	Argument description
onFieldChanged	Fired after a field has changed. If the field change ran rules which updated other fields, all these changes are part of a single event.	ID	The ID of the work item.
changedFields	Array with the reference name of all changed fields.	ID	The ID of the work item.
onLoaded	Fired after when the data gets loaded in the work item form, when the user opens a work item, or when the user navigates to another work item in the triage view.	ID	The ID of the work item.
onReset	Fired after the user undoes the changes to the work item.	ID	The ID of the work item.
onRefreshed	Fired after the user refreshed the work item manually.	ID	The ID of the work item.
onSaved	Fired after a work item is saved. For work items in a dialog, you should target the "ms.vss-work-web.work-item-notifications" type to ensure the event fires since once the dialog closes, this contribution type gets unloaded. For more information, see Listen for events.	ID	The ID of the work item.
onUnloaded	Fired before the user closes the dialog, or before the user moves to another work item in the triage view.	ID	The ID of the work item.

Add a page

A new page is rendered as a tab on the work item form. New pages appear next to the Details tab.



To add a page to the work item form, add a contribution to your extension manifest. The type of this contribution should be `ms.vss-work-web.work-item-form-page` and it should

target the `ms.vss-work-web.work-item-form` contribution.

JSON

```
"contributions": [
    {
        "id": "sample-work-item-form-page",
        "type": "ms.vss-work-web.work-item-form-page",
        "description": "Custom work item form page",
        "targets": [
            "ms.vss-work-web.work-item-form"
        ],
        "properties": {
            "name": "My Page",
            "uri": "workItemPage.html"
        }
    }
]
```

Properties

Expand table

Property	Description
name	Text that appears on the tab page.
uri	URI to a page that hosts the html that shows on the work item form and its scripts.

JavaScript sample

See the JavaScript sample in the form group section. The name of the registered object should match the `id` of the contribution.

Events

Expand table

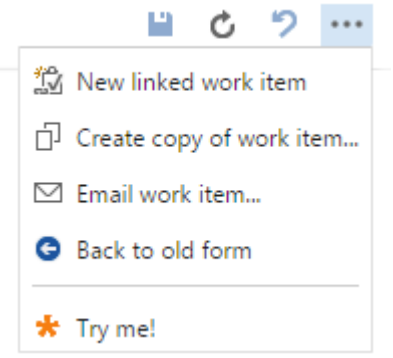
Event	Event description	Argument	Argument description
onFieldChanged	Fired after a field has changed. If the field change ran rules which updated other fields, all these changes are part of a single event.	ID	The ID of the work item.

Event	Event description	Argument	Argument description
changedFields	Array with the reference name of all changed fields.	ID	The ID of the work item.
onLoaded	Fired after when the data gets loaded in the work item form, when the user opens a work item, or when the user navigates to another work item in the triage view.	ID	The ID of the work item.
onReset	Fired after the user undoes the changes to the work item.	ID	The ID of the work item.
onRefreshed	Fired after the user refreshed the work item manually.	ID	The ID of the work item.
onSaved	Fired after a work item is saved. For work items in a dialog, you should target the "ms.vss-work-web.work-item-notifications" type to ensure the event fires since once the dialog closes, this contribution type gets unloaded. For more information, see Listen for events.	ID	The ID of the work item.
onUnloaded	Fired before the user closes the dialog, or before the user moves to another work item in the triage view.	ID	The ID of the work item.

Configure contributions in work item form

In Azure DevOps Services, by default the group extensions appear in the end of the second column of the form and page contributions appear after all the work item form pages as a tab. Control contributions are not shown in the form by default so users have to manually add them to the form. In Azure DevOps Server, to show/hide or move the control, group and page contributions in work item form, see Configure work item form extensions.

Add menu action



To add an item to the work item toolbar, add this contribution to your extension manifest. The item appears in the ... dropdown in the top right of the work item form.

JSON

```
"contributions": [
   {
      "id": "sample-work-item-menu",
      "type": "ms.vss-web.action",
      "description": "Sample toolbar item which updates the title of a work
item",
      "targets": [
          "ms.vss-work-web.work-item-context-menu"
      ],
      "properties": {
          "text": "Try me!",
          "title": "Updates the title of the work item from the extension",
          "toolbarText": "Try me!",
          "icon": "images/show-properties.png",
          "uri": "menu-workItemToolbarButton.html"
      }
   }
]
```

Properties

Expand table

Property	Description
text	Text that appears on the toolbar item.
title	Tooltip text that appears on the menu item.
toolbarText	Text that appears when the item is being hovered over.
uri	URI to a page that registers the toolbar action handler.

Property	Description
icon	URL to an icon that appears on the menu item. Relative URLs are resolved using baseUri.
group	Determines where the menu item appears, related to others. Toolbar items with the same group name are grouped and divided by a separator from the rest of the items.
registeredObjectId	(Optional) Name of the registered menu action handler. Defaults to the contribution ID.

Listen for events

To add an observer to the work item, which listens to the work item events, add this contribution to your extension manifest. There's no visualization for observers on the work item form. This is the best way to listen to work item form onSaved event since the observer lives outside of the form and doesn't get destroyed when form closes, which might happen right after save.

JSON

```
"contributions": [
   {
       "id": "sample-work-item-form-observer",
       "type": "ms.vss-work-web.work-item-notifications",
       "description": "Gets events about the current work item form for the
'Try Me!' toolbar button",
       "targets": [
           "ms.vss-work-web.work-item-form"
       ],
       "properties": {
           "uri": "myformobserver.html"
       }
   }
]
```

Properties

Expand table

Property	Description
uri	URI to a page that hosts the scripts listening to events.

Events

Expand table

Event	Event description	Argument	Argument description
onFieldChanged	Fired after a field has changed. If the field change ran rules which updated other fields, all these changes are part of a single event.	ID	The ID of the work item.
changedFields	Array with the reference name of all changed fields.	ID	The ID of the work item.
onLoaded	Fired after when the data gets loaded in the work item form, when the user opens a work item, or when the user navigates to another work item in the triage view.	ID	The ID of the work item.
onReset	Fired after the user undoes the changes to the work item.	ID	The ID of the work item.
onRefreshed	Fired after the user refreshed the work item manually.	ID	The ID of the work item.
onSaved	Fired after a work item is saved. For work items in a dialog, you should target the "ms.vss-work-web.work-item-notifications" type to ensure the event fires since once the dialog closes, this contribution type gets unloaded. For more information, see Listen for events.	ID	The ID of the work item.
onUnloaded	Fired before the user closes the dialog, or before the user moves to another work item in the triage view.	ID	The ID of the work item.

HTML/JavaScript sample

HTML

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Work item extension page sample</title>
</head>

<body>
```

```
    <script src="sdk/scripts/SDK.js"></script>

    <script>
        SDK.init({
            usePlatformScripts: true
        });

        SDK.ready(function () {
             // Register a listener for the work item page contribution.
            SDK.register(SDK.getContributionId(), function () {
                return {
                    // Called when the active work item is modified
                    onFieldChanged: function(args) {

                    },

                    // Called when a new work item is being loaded in the UI
                    onLoaded: function (args) {

                    },

                    // Called when the active work item is being unloaded in
the UI
                    onUnloaded: function (args) {

                    },

                    // Called after the work item has been saved
                    onSaved: function (args) {

                    },

                    // Called when the work item is reset to its unmodified
state (undo)
                    onReset: function (args) {

                    },

                    // Called when the work item has been refreshed from the
server
                    onRefreshed: function (args) {

                    }
                }
            });
        });
     </script>
</body>
</html>
```

Changes with New Boards Hub

> **Note**
>
> New Boards Hub is currently in preview but is on track be available for everyone in 2024. We strongly suggest that you immediately enable New Boards Hub and test your internal extensions. Learn more.

Use the latests SDK's

Make sure your extension is using the latest version of azure-devops-extension-sdk

When using the new SDK, you should also be using the azure-devops-extension-api package for REST APIs. We update the methods and interfaces every sprint to ensure it includes all of the latest features.

When to use action or action-provider

Use `ms.vss-web.action-provider` when dynamically loading menu items using `getMenuItems` on the menu handler. Avoid using `ms.vss-web.action-provider` when your menu items are static and defined in your manifest. Instead `ms.vss-web.action` should be used .

Package require("VSS/Events/Document") is no longer supported

`require("VSS/Events/Document")` import is no longer supported with New Boards Hub.

Feedback

Was this page helpful? Yes No

Provide product feedback

Add a custom control to the work item form

Article • 09/09/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Custom controls allow you to change how users view and interact with a field on the work item form. The following article walks you through how this sample custom control was built. This article shows you how to build your own custom control.

> **Tip**
>
> Check out our newest documentation on extension development using the Azure DevOps Extension SDK.

Prerequisites

Include the azure-devops-extension-sdk in your project:

1. Install the SDK via npm: `npm install azure-devops-extension-sdk`.
2. Initialize it in your JavaScript code:

```JavaScript
import * as SDK from "azure-devops-extension-sdk";
SDK.init();
```

Add the custom control

To add a control to the main page, add a contribution to your extension manifest. The type of contribution should be `ms.vss-work-web.work-item-form-control` and it should target the `ms.vss-work-web.work-item-form` contribution.

```JSON
"contributions": [
    {
        "id": "sample-work-item-form-control",
        "type": "ms.vss-work-web.work-item-form-control",
```

```
        "description": "Custom work item form control",
        "targets": [
            "ms.vss-work-web.work-item-form"
        ],
        "properties": {
            "name": "My Control",
            "uri": "workItemControl.html",
            "height": 600
        }
    }
],
"manifestVersion": 1.0,
    "scopes": [
        "vso.work"
    ]
```

The work item form adds an IFrame to host the custom control.

Properties

Expand table

Property	Description
`name`	Text that appears on the group.
`uri`	URI to a page that hosts the html that is loaded by the IFrame.
`height`	(Optional) Defines the height of the IFrame. When omitted, it's 50 pixels.
`inputs`	The values a user provides within the form.

If you want to dynamically resize the IFrame, you can use the `resize method` available in the client SDK.

A custom control on the work item form is another type of contribution, like group & page contribution. The main difference is that a control contribution can take a set of user inputs while group and page contributions can't.

Control contribution inputs

To define the inputs for your control contribution, use the `inputs` property in the contribution object in the manifest.

In the following sample you see two inputs: `FieldName` and `Colors`. `FieldName` specifies with which field the control associates. `Colors` configures which colors map to which

values in the control.

The values for the inputs get provided by the users when they add to the work item form. These values get passed to the control contribution when it's loaded on the form.

JSON

```
"inputs": [
    {
        "id": "FieldName",
        "description": "The field associated with the control.",
        "type": "WorkItemField",
        "properties": {
            "workItemFieldTypes": ["String"]
        },
        "validation": {
            "dataType": "String",
            "isRequired": true
        }
    },
    {
        "id": "Colors",
        "description": "The colors that match the values in the control.",
        "type": "string",
        "validation": {
            "dataType": "String",
            "isRequired": false
        }
    }
]
```

The following properties define a user input that the contribution can use:

- **id** - A unique ID for the input.
- **description** - A few sentences describing the input.
- **type (optional)** - The type of input.
  - Valid values:
    - `WorkItemField` - Indicates that the input is a Work Item field. The value provided by the user for this input should be a reference name for the valid work item field.
- **properties (optional)** - Custom properties for the input.
  - Valid keys:
    - `workItemFieldTypes` - Defines an array of field types that this input supports. Valid values:
      - `String`
      - `Integer`
      - `DateTime`

        - `PlainText`
        - `HTML`
        - `TreePath`
        - `History`
        - `Double`
        - `Guid`
        - `Boolean`
        - `Identity`
        - `PicklistString`
        - `PicklistInteger`
        - `PicklistDouble`
- **validation** - A set of properties that defines which values are considered valid for the input.
    - Valid keys:
        - `dataType` - Defines the data type of the input value. Valid values for this property:
            - `String`
            - `Number`
            - `Boolean`
            - `Field`
        - `isRequired` - A boolean value, which indicates whether the input is required to have a value.

JavaScript sample

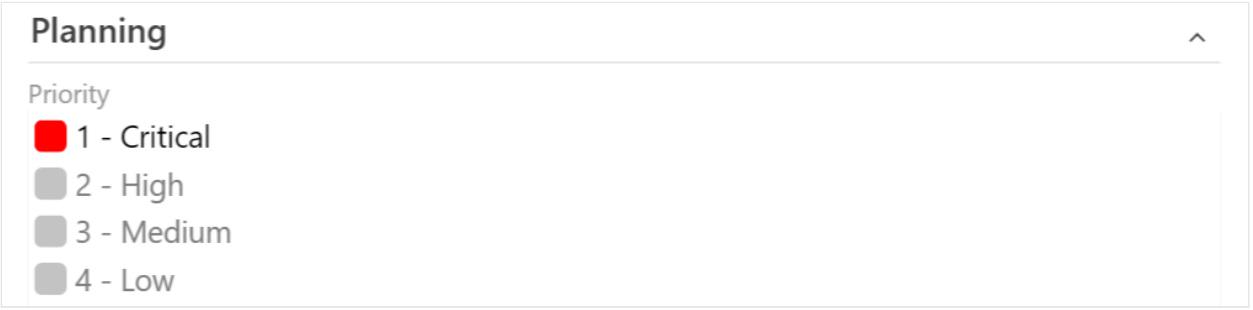
A control extension works like a group or page extension with one difference that it can take certain user inputs. To read the user input values, use `VSS.getConfiguration().witInputs`. The following sample shows how to register an object that's called when events occur on the work item form that may affect your contributed control. It also shows how to read input values provided by user for this control.

TypeScript

```
import { Control } from "control";
import * as SDK from "azure-devops-extension-sdk";
import * as ExtensionContracts from "azure-devops-extension-
api/WorkItemTracking/WorkItemTracking";

let control;
```

```
const provider = () => {
    return {
        onLoaded: (workItemLoadedArgs) => {
            // create the control
            const config = SDK.getConfiguration();
            const fieldName = config.witInputs["FieldName"];
            const colors = config.witInputs["Colors"];
            control = new Control(fieldName, colors);
        },
        onFieldChanged: (fieldChangedArgs) => {
            const changedValue =
fieldChangedArgs.changedFields[control.getFieldName()];
            if (changedValue !== undefined) {
                control.updateExternal(changedValue);
            }
        }
    };
};

SDK.init();
SDK.ready().then(() => {
    SDK.register(SDK.getContributionId(), provider);
});
```

The following screenshot shows a sample custom work item control for the *Priority* field.



Feedback

Was this page helpful? Yes No

Provide product feedback

Add extensions in work item form via work item type definition xml

Article • 02/06/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

You can export a work item type as xml using the `witadmin` tool, which includes the layout of the work item form. As part of this example, we add the page, group, and control contributions to the layout. We also add the control to the Agile "User Story" work item type. For more information, see WebLayout xml reference.

> **Note**
>
> Work item form customization via xml is supported only on Azure DevOps Server, not Azure DevOps Services.

Add extension in work item form

1. Install work item form extensions in Azure DevOps Server.
2. Open the `Developer Command Prompt`. Export the xml file to your desktop with the following command.

   ```
   witadmin exportwitd /collection:CollectionURL /p:Project /n:TypeName
   /f:FileName
   ```

   A file gets created in the directory that you specified.
3. Inside this file, go to the **WebLayout** section. Inside the WebLayout section, a comment blob specifies which installed extensions target work item forms for your collection. For each extension, all its form contributions are listed with their IDs and inputs (if it's a Control contribution). In the following example, the comment shows the *color-control-dev* extension installed on the collection. The extension has one control contribution that takes two inputs.

   ```
   XML
   ```

```
    <!--**********************************Work Item
Extensions*************************

 Extension:
    Name: color-control-dev
    Id: example.color-control-dev

    Control contribution:
        Id: example.color-control-dev.color-control-contribution
        Description:
        Inputs:
            Id: FieldName
            Description: The field associated with the control.
            Type: Field
            IsRequired: true

            Id: Colors
            Descriptions: The colors that match the values in the
control.
            Type: String
            IsRequired: false
```

4. Find your extension ID in the **Work Item Extensions** section:

XML

```
    <!--**********************************Work Item
Extensions*************************

 Extension:
    Name: color-control-dev
    Id: example.color-control-dev
    ...
```

5. Add an extension tag below the **Work Item Extensions** section, shown as follows, to make your extension available to the work item form. To place a contribution inside the form, its extension must be specified in the `Extensions` section.

XML

```
    <!--**********************************Work Item
Extensions*************************
    ...

    Note: For more information on work item extensions use the
following topic:
    https://go.microsoft.com/fwlink/?LinkId=816513
    -->

    <Extensions>
```

```
        <Extension Id="example.color-control-dev" />
    </Extensions>
```

6. Specifying the extensions in the xml automatically places both the **page** and **group** contributions defined in the extensions inside the form. You can move the contributions in the following examples.

Add page contribution

XML

```
<Page Id="Details">
<PageContribution Id="<page contribution id>" />
...
```

Add group contribution

XML

```
<Page Id="Details">
...
    <Section>
    ...
        <GroupContribution Id="<group contribution id>" />
        ...
```

A page contribution and a group contribution can't take any other layout elements.

Add control contribution

Unlike *page* and *group* contributions, specifying the extensions in the xml doesn't automatically place *control* contributions. To add these contributions in the form, add them with a contribution tag inside the form. The following example adds the *ControlContribution* to the *Planning* group.

If a control contribution has any required input defined, users must give a value for that input. For any non-required input, users can decide whether to set a value to the input. In the following example, the `FieldName` and `Colors` inputs get set.

XML

```
<Page Id="Details">
...
```

```
        <Section>
        ...
            <Group Id="Planning">
            ...
                <ControlContribution Label="Priority" Id="example.color-
control-dev.color-control-contribution">
                    <Inputs>
                        <Input Id="FieldName" Value="Microsoft.Azure DevOps
Services.Common.Priority" />
                        <Input Id="Colors" Value="red;green" />
                    </Inputs>
                </ControlContribution>

                <Control Label="Risk" Type="FieldControl"
FieldName="Microsoft.Azure DevOps Services.Common.Risk" />
```

7. Import this xml file, using `witadmin`.

```
witadmin importwitd /collection:CollectionURL /p:Project /f:FileName
```

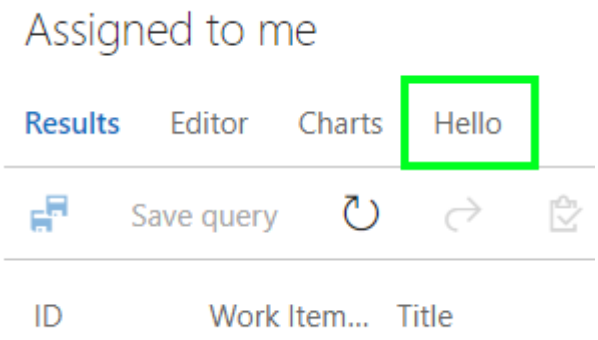
Your extension is configured via the work item form!

Add tabs on query result pages

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

If you have a web page that can be hosted in an iframe, it can be hosted in Azure DevOps as a tab on the query result pages. In this example, we'll add a simple Hello World tab on query results.



> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Create your web page

1. Get the Client SDK `SDK.js` file and add it to your web app. Place it in the `home/sdk/scripts` folder.
   a. Use the 'npm install' command to retrieve the SDK: `npm install azure-devops-extension-sdk`.
   b. To learn more about the SDK, visit the Client SDK GitHub Page.
2. Add the web page that you want to display as a hub. We're doing a simple `hello-world.html` page here, added to the `home` directory.
3. In your HTML page, add a reference to the SDK, and call `init()` and `notifyLoadSucceeded()`.

```
HTML
```

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
   <title>Hello World</title>
   <script src="sdk/scripts/SDK.js"></script>
</head>
<body>
   <script type="text/javascript">SDK.init();</script>
   <h1>Hello World</h1>
   <script type="text/javascript">SDK.notifyLoadSucceeded();</script>
</body>
</html>
```

Update your extension manifest

Update your extension manifest file with the following code:

JSON

```
...

    "contributions": [
        {
            "id": "Fabrikam.HelloTab.Query.Tabs",
            "type": "ms.vss-web.tab",
            "description": "Adds a 'Hello' tab to the query results",
            "targets": [
                "ms.vss-work-web.query-tabs"
            ],
            "properties": {
                "uri": "hello-query-tab.html",
                "title": "Hello",
                "name": "Hello"
            }
        }
    ],
    "scopes": [
        "vso.work"
    ],
    "files": [
        {
            "path": "hello-query-tab.html", "addressable": true
        },
        {
            "path": "scripts", "addressable": true
        },
        {
            "path": "sdk/scripts", "addressable": true
        },
        {
            "path": "images/logo.png", "addressable": true
```

```
        }
    ]
...
```

Contributions

The **contributions** stanza contains information about your tasks.

For each contribution in your extension, the manifest defines

- the type of contribution (tab in this case),
- the contribution target (the query tabs in this case),
- and the properties that are specific to each type of contribution. For a tab, we have

Property	Description
name	Name of the tab
uri	Path (relative to the extension's base URI) of the page to surface as the tab
title	Title of the tab to display

Scopes

It includes the scopes that your extension requires. In this case, we need `vso.work` to access work items.

Files

Include all of the files your extension accesses.
For your files, set `addressable` to `true` unless you include other files that don't need to be URL-addressable.

Example

JavaScript

```
SDK.register(SDK.getContributionId(), {
    pageTitle: function(state) {
        return "Hello";
    },
    updateContext: function(tabContext) {
    },
    isInvisible: function(state) {
```

```
            return false;
        },
        isDisabled: function(state) {
            return false;
        }
    });
```

Learn about all of the places where you can add a hub in Extensibility points.

Next steps

Package, Publish, and Install or

Test and Debug

Add a dashboard widget

Article • 08/06/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Widgets on a dashboard are implemented as contributions in the extension framework. A single extension can have multiple contributions. Learn how to create an extension with multiple widgets as contributions.

This article is divided into three parts, each building on the previous - beginning with a simple widget and ending with a comprehensive widget.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Prerequisites

- **Knowledge:** Some knowledge of JavaScript, HTML, CSS is required for widget development.
- An **organization** in Azure DevOps.
- A **text editor**. For many of the tutorials, we use Visual Studio Code.
- The latest version of **node**.
- **Cross-platform CLI for Azure DevOps (tfx-cli)** to package your extensions.
  - **tfx-cli** can be installed using `npm`, a component of Node.js by running `npm i -g tfx-cli`
- A home directory for your project. This directory is referred to as `home` throughout the tutorial.

Extension file structure:

no-highlight

```
|--- README.md
|--- sdk
    |--- node_modules
    |--- scripts
        |--- VSS.SDK.min.js
|--- img
    |--- logo.png
|--- scripts
```

```
|--- hello-world.html               // html page to be used for your widget
|--- vss-extension.json             // extension's manifest
```

In this tutorial

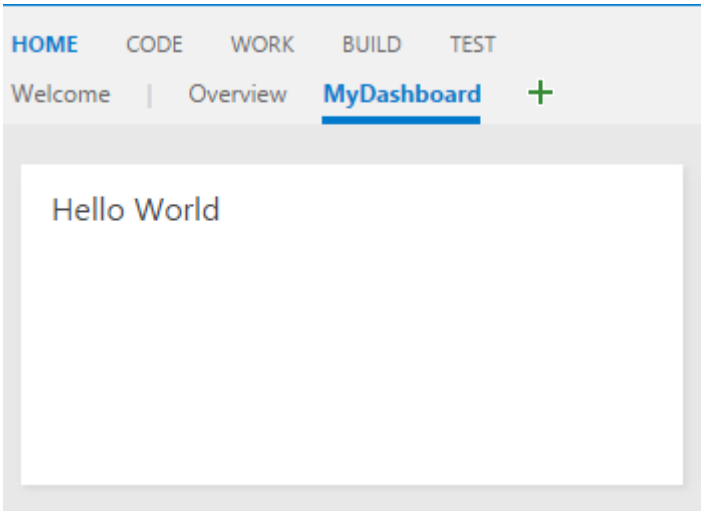
1. Part 1: Shows you how to create a new widget, which prints a simple "Hello World" message.
2. Part 2: Builds on the first part by adding a call to an Azure DevOps REST API.
3. Part 3: Explains how to add configuration to your widget.

> **Note**
>
> If you're in a hurry and want to get your hands on the code right away, you can download the samples. Once downloaded, go to the `widgets` folder, then follow Step 6 and Step 7 directly to publish the sample extension which has the three sample widgets of varying complexities.

Get started with some basic styles for widgets that we provide out-of-the-box and some guidance on widget structure.

Part 1: Hello World

Part 1 presents a widget that prints "Hello World" using JavaScript.



Step 1: Get the client SDK - `VSS.SDK.min.js`

The core SDK script, `VSS.SDK.min.js`, enables web extensions to communicate to the host Azure DevOps frame. The script does operations like initializing, notifying extension

is loaded, or getting context about the current page. Get the Client SDK `VSS.SDK.min.js` file and add it to your web app. Place it in the `home/sdk/scripts` folder.

To retrieve the SDK, use the 'npm install' command:

```plaintext
npm install vss-web-extension-sdk
```

For more information, see the Client SDK GitHub page.

Step 2: Set up your HTML page - `hello-world.html`

Your HTML page is the glue that holds your layout together and includes references to CSS and JavaScript. You can name this file anything. Update all references to `hello-world` with the name you use.

Your widget is HTML based and is hosted in an iframe. Add the following HTML in `hello-world.html`. We add the mandatory reference to `VSS.SDK.min.js` file and include an `h2` element, which is updated with the string Hello World in the upcoming step.

```HTML
<!DOCTYPE html>
<html>
    <head>
        <script src="sdk/scripts/VSS.SDK.min.js"></script>
    </head>
    <body>
        <div class="widget">
            <h2 class="title"></h2>
        </div>
    </body>
</html>
```

Even though we're using an HTML file, most of the HTML head elements other than script and link are ignored by the framework.

Step 3: Update JavaScript

We use JavaScript to render content in the widget. In this article, we wrap all of our JavaScript code inside a `<script>` element in the HTML file. You can choose to have this code in a separate JavaScript file and refer it in the HTML file. The code renders the content. This JavaScript code also initializes the VSS SDK, maps the code for your

widget to your widget name, and notifies the extension framework of widget successes or failures. In our case, the following code prints "Hello World" in the widget. Add this `script` element in the `head` of the HTML.

HTML

```
<script type="text/javascript">
    VSS.init({
        explicitNotifyLoaded: true,
        usePlatformStyles: true
    });

    VSS.require("TFS/Dashboards/WidgetHelpers", function (WidgetHelpers) {
        WidgetHelpers.IncludeWidgetStyles();
        VSS.register("HelloWorldWidget", function () {
            return {
                load: function (widgetSettings) {
                    var $title = $('h2.title');
                    $title.text('Hello World');

                    return WidgetHelpers.WidgetStatusHelper.Success();
                }
            };
        });
        VSS.notifyLoadSucceeded();
    });
</script>
```

- `VSS.init` initializes the handshake between the iframe hosting the widget and the host frame.
- We pass `explicitNotifyLoaded: true` so that the widget can explicitly notify the host when it's done loading. This control allows us to notify load completion after ensuring that the dependent modules are loaded. We pass `usePlatformStyles: true` so that the widget can use Azure DevOps core styles for HTML elements (such as body, div, and so on). If the widget prefers to not use these styles, they can pass in `usePlatformStyles: false`.
- `VSS.require` is used to load the required VSS script libraries. A call to this method automatically loads general libraries like JQuery and JQueryUI. In our case, we depend on the WidgetHelpers library, which is used to communicate widget status to the widget framework. So, we pass the corresponding module name `TFS/Dashboards/WidgetHelpers` and a callback to `VSS.require`. The callback is called once the module is loaded. The callback has the rest of the JavaScript code needed for the widget. At the end of the callback, we call `VSS.notifyLoadSucceeded` to notify load completion.

- `WidgetHelpers.IncludeWidgetStyles` includes a stylesheet with some basic css to get you started. To make use of these styles, wrap your content inside an HTML element with class `widget`.
- `VSS.register` is used to map a function in JavaScript, which uniquely identifies the widget among the different contributions in your extension. The name should match the `id` that identifies your contribution as described in Step 5. For widgets, the function that is passed to `VSS.register` should return an object that satisfies the `IWidget` contract, for example, the returned object should have a load property whose value is another function that has the core logic to render the widget. In our case, it's to update the text of the `h2` element to "Hello World." It's this function that is called when the widget framework instantiates your widget. We use the `WidgetStatusHelper` from WidgetHelpers to return the `WidgetStatus` as success.

> ⚠ **Warning**
>
> If the name used to register the widget doesn't match the ID for the contribution in the manifest, then the widget functions unexpectedly.

- `vss-extension.json` should always be at the root of the folder (in this guide, `HelloWorld`). For all the other files, you can place them in whatever structure you want inside the folder, just make sure to update the references appropriately in the HTML files and in the `vss-extension.json` manifest.

Step 4: Update your extension logo: `logo.png`

Your logo is displayed in the Marketplace and in the widget catalog once a user installs your extension.

You need a 98 px x 98-px catalog icon. Choose an image, name it `logo.png`, and place it in the `img` folder.

You can name these images however you want as long as the extension manifest in the next step is updated with the names you use.

Step 5: Create your extension manifest: `vss-extension.json`

Every extension must have an extension manifest file.

- Read the extension manifest reference.

- Find out more about the contribution points in Extensibility points.
- Create a json file (`vss-extension.json`, for example) in the `home` directory with the following contents:

JSON

```
{
    "manifestVersion": 1,
    "id": "azure-devops-extensions-myExtensions",
    "version": "1.0.0",
    "name": "My First Set of Widgets",
    "description": "Samples containing different widgets extending
dashboards",
    "publisher": "fabrikam",
    "categories": ["Azure Boards"],
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "icons": {
        "default": "img/logo.png"
    },
    "contributions": [
        {
            "id": "HelloWorldWidget",
            "type": "ms.vss-dashboards-web.widget",
            "targets": [
                "ms.vss-dashboards-web.widget-catalog"
            ],
            "properties": {
                "name": "Hello World Widget",
                "description": "My first widget",
                "catalogIconUrl": "img/CatalogIcon.png",
                "previewImageUrl": "img/preview.png",
                "uri": "hello-world.html",
                "supportedSizes": [
                    {
                        "rowSpan": 1,
                        "columnSpan": 2
                    }
                ],
                "supportedScopes": ["project_team"]
            }
        }
    ],
    "files": [
        {
            "path": "hello-world.html",
            "addressable": true
        },
        {
            "path": "sdk/scripts",
```

```
            "addressable": true
        },
        {
            "path": "img",
            "addressable": true
        }
    ]
}
```

For more information about required attributes, see the Extension manifest reference.

> ⓘ Note
>
> Change the **publisher** to your publisher name. To create a publisher, see Package/Publish/Install.

Icons

The **icons** stanza specifies the path to your extension's icon in your manifest.

Contributions

Each contribution entry defines properties.

- The **ID** to identify your contribution. This ID should be unique within an extension. This ID should match with the name you used in Step 3 to register your widget.
- The **type** of contribution. For all widgets, the type should be `ms.vss-dashboards-web.widget`.
- The array of **targets** to which the contribution is contributing. For all widgets, the target should be `[ms.vss-dashboards-web.widget-catalog]`.
- The **properties** are objects that include properties for the contribution type. For widgets, the following properties are mandatory.

Expand table

Property	Description
name	Name of the widget to display in the widget catalog.
description	Description of the widget to display in the widget catalog.
catalogIconUrl	Relative path of the catalog icon that you added in Step 4 to display in the widget catalog. The image should be 98 px x 98 px. If you used a different

Property	Description
	folder structure or a different file name, then specify the appropriate relative path here.
previewImageUrl	Relative path of the preview image that you added in Step 4 to display in the widget catalog. The image should be 330 px x 160 px. If you used a different folder structure or a different file name, then specify the appropriate relative path here.
uri	Relative path of the HTML file that you added in Step 1. If you used a different folder structure or a different file name, then specify the appropriate relative path here.
supportedSizes	Array of sizes supported by your widget. When a widget supports multiple sizes, the first size in the array is the default size of the widget. The `widget size` is specified for the rows and columns occupied by the widget in the dashboard grid. One row/column corresponds to 160 px. Any dimension larger than 1x1 gets an extra 10 px that represent the gutter between widgets. For example, a 3x2 widget is `160*3+10*2` wide and `160*2+10*1` tall. The maximum supported size is `4x4`.
supportedScopes	Currently, only team dashboards are supported. The value has to be `project_team`. Future updates might include more options for dashboard scopes.

Files

The **files** stanza states the files that you want to include in your package - your HTML page, your scripts, the SDK script, and your logo. Set `addressable` to `true` unless you include other files that don't need to be URL-addressable.

> ⓘ **Note**
>
> For more information about the **extension manifest file**, such as its properties and what they do, check out the extension manifest reference.

Step 6: Package, publish, and share

Once you have your written extension, the next step towards getting it into the Marketplace is to package all of your files together. All extensions are packaged as VSIX 2.0 compatible .vsix files - Microsoft provides a cross-platform command line interface (CLI) to package your extension.

Get the packaging tool

You can install or update the Cross-platform CLI for Azure DevOps (tfx-cli) using `npm`, a component of Node.js, from your command line.

```no-highlight
npm i -g tfx-cli
```

Package your extension

Packaging your extension into a .vsix file is effortless once you have the tfx-cli. Go to your extension's home directory and run the following command.

```no-highlight
tfx extension create --manifest-globs vss-extension.json
```

> **Note**
>
> An extension/integration version must be incremented on every update.
> When you update an existing extension, either update the version in the manifest or pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and save the new version to your manifest.

After you package your extension in a .vsix file, you're ready to publish your extension to the Marketplace.

Create publisher for the extension

All extensions, including extensions from Microsoft, are identified as being provided by a publisher. If you aren't already a member of an existing publisher, create one.

1. Sign in to the Visual Studio Marketplace Publishing Portal
2. If you aren't already a member of an existing publisher, you must create a publisher. If you already have a publisher, scroll to and select **Publish Extensions** under **Related Sites**.

    - Specify an identifier for your publisher, for example: `mycompany-myteam`
        - The identifier is used as the value for the `publisher` attribute in your extensions' manifest file.

- Specify a display name for your publisher, for example: `My Team`

3. Review the Marketplace Publisher Agreement and select **Create**.

Now your publisher is defined. In a future release, you can grant permissions to view and manage your publisher's extensions.

Publishing extensions under a common publisher simplifies the process for teams and organizations, offering a more secure approach. This method eliminates the need to distribute a single set of credentials among multiple users, enhancing security and

Update the `vss-extension.json` manifest file in the samples to replace the dummy publisher ID `fabrikam` with your publisher ID.

Publish and share the extension

Now, you can now upload your extension to the Marketplace.

Select **Upload new extension**, go to your packaged .vsix file, and select **Upload**.

You can also upload your extension via the command line by using the `tfx extension publish` command instead of `tfx extension create` to package and publish your extension in one step. You can optionally use `--share-with` to share your extension with one or more accounts after publishing. You also need a personal access token.

```no-highlight
tfx extension publish --manifest-globs your-manifest.json --share-with
yourOrganization
```

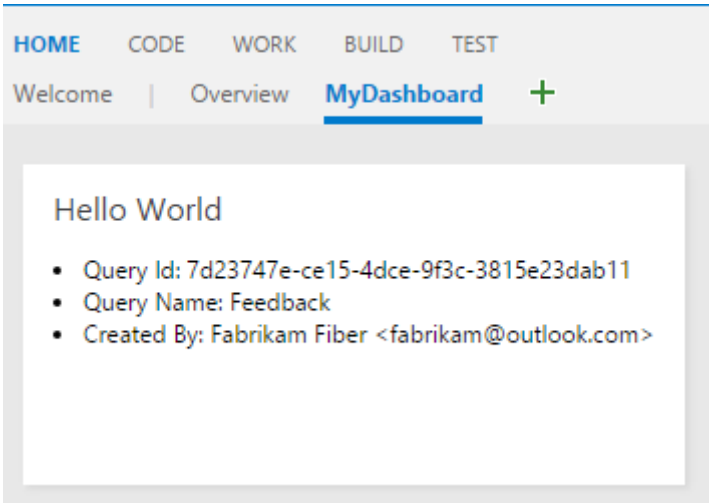
Step 7: Add widget from the catalog

1. Sign in to your project, `http://dev.azure.com/{Your_Organization}/{Your_Project}`.
2. Select **Overview** > **Dashboards**.
3. Select **Add a widget**.
4. Highlight your widget, and then select **Add**.

   The widget appears on your dashboard.

Part 2: Hello World with Azure DevOps REST API

Widgets can call any of the REST APIs in Azure DevOps to interact with Azure DevOps resources. In the following example, we use the REST API for WorkItemTracking to fetch information about an existing query and display some query info in the widget under the "Hello World" text.



Step 1: Add HTML file

Copy the file `hello-world.html` from the previous example, and rename the copy to `hello-world2.html`. Your folder now looks like the following example:

|--- README.md |--- node_modules
|--- SDK
|--- scripts |--- VSS.SDK.min.js
|--- img
|--- logo.png
|--- scripts
|--- hello-world.html // html page to be used for your widget
|--- hello-world2.html // renamed copy of hello-world.html |--- vss-extension.json // extension's manifest

To hold the query information, add a new `div` element under the `h2`. Update the name of the widget from `HelloWorldWidget` to `HelloWorldWidget2` in the line where you call `VSS.register`. This action allows the framework to uniquely identify the widget within the extension.

```
HTML
```

```
<!DOCTYPE html>
<html>
    <head>
        <script src="sdk/scripts/VSS.SDK.min.js"></script>
        <script type="text/javascript">
            VSS.init({
                explicitNotifyLoaded: true,
                usePlatformStyles: true
            });

            VSS.require("TFS/Dashboards/WidgetHelpers", function
(WidgetHelpers) {
                WidgetHelpers.IncludeWidgetStyles();
                VSS.register("HelloWorldWidget2", function () {
                    return {
                        load: function (widgetSettings) {
                            var $title = $('h2.title');
                            $title.text('Hello World');

                            return
WidgetHelpers.WidgetStatusHelper.Success();
                        }
                    }
                });
                VSS.notifyLoadSucceeded();
            });
        </script>
    </head>
    <body>
        <div class="widget">
            <h2 class="title"></h2>
            <div id="query-info-container"></div>
        </div>
    </body>
</html>
```

Step 2: Access Azure DevOps resources

To enable access to Azure DevOps resources, scopes need to be specified in the extension manifest. We add the `vso.work` scope to our manifest.

This scope indicates the widget needs read-only access to queries and work items. See all available scopes here. Add the following code at the end of your extension manifest.

JSON

```
{
    "scopes":[
        "vso.work"
```

```
    ]
}
```

To include other properties, you should list them explicitly, for example:

JSON

```
{
    "name": "example-widget",
    "publisher": "example-publisher",
    "version": "1.0.0",
    "scopes": [
        "vso.work"
    ]
}
```

> ⚠ **Warning**
>
> Adding or changing scopes after publishing an extension is currently not supported. If you already uploaded your extension, remove it from the Marketplace. Go to the Visual Studio Marketplace Publishing Portal, right-select your extension and select **Remove**.

Step 3: Make the REST API Call

There are many client-side libraries that can be accessed via the SDK to make REST API calls in Azure DevOps. These libraries are called REST clients and are JavaScript wrappers around Ajax calls for all available server-side endpoints. You can use methods provided by these clients instead of writing Ajax calls yourself. These methods map the API responses to objects that your code can consume.

In this step, we update the `VSS.require` call to load `AzureDevOps/WorkItemTracking/RestClient`, which provides the WorkItemTracking REST client. We can use this REST client to get information about a query called `Feedback` under the folder `Shared Queries`.

Inside the function that we pass to `VSS.register`, we create a variable to hold the current project ID. We need this variable to fetch the query. We also create a new method `getQueryInfo` to use the REST client. This method that is then called from the load method.

The method `getClient` gives an instance of the REST client we need. The method `getQuery` returns the query wrapped in a promise. The updated `VSS.require` looks as follows:

JavaScript

```
VSS.require(["AzureDevOps/Dashboards/WidgetHelpers",
"AzureDevOps/WorkItemTracking/RestClient"],
    function (WidgetHelpers, TFS_Wit_WebApi) {
        WidgetHelpers.IncludeWidgetStyles();
        VSS.register("HelloWorldWidget2", function () {
            var projectId = VSS.getWebContext().project.id;

            var getQueryInfo = function (widgetSettings) {
                // Get a WIT client to make REST calls to Azure DevOps
Services
                return TFS_Wit_WebApi.getClient().getQuery(projectId,
"Shared Queries/Feedback")
                    .then(function (query) {
                        // Do something with the query

                        return WidgetHelpers.WidgetStatusHelper.Success();
                    }, function (error) {
                        return
WidgetHelpers.WidgetStatusHelper.Failure(error.message);
                    });
            }

            return {
                load: function (widgetSettings) {
                    // Set your title
                    var $title = $('h2.title');
                    $title.text('Hello World');

                    return getQueryInfo(widgetSettings);
                }
            }
        });
        VSS.notifyLoadSucceeded();
    });
```

Notice the use of the Failure method from `WidgetStatusHelper`. It allows you to indicate to the widget framework that an error occurred and take advantage to the standard error experience provided to all widgets.

If you don't have the `Feedback` query under the `Shared Queries` folder, then replace `Shared Queries\Feedback` in the code with the path of a query that exists in your project.

Step 4: Display the response

The last step is to render the query information inside the widget. The `getQuery` function returns an object of type `Contracts.QueryHierarchyItem` inside a promise. In this example, we display the query ID, the query name, and the name of the query creator under the "Hello World" text. Replace the `// Do something with the query` comment with the following code:

JavaScript

```
// Create a list with query details
var $list = $('<ul>');
$list.append($('<li>').text("Query Id: " + query.id));
$list.append($('<li>').text("Query Name: " + query.name));
$list.append($('<li>').text("Created By: " + (query.createdBy ?
query.createdBy.displayName : "<unknown>")));

// Append the list to the query-info-container
var $container = $('#query-info-container');
$container.empty();
$container.append($list);
```

Your final `hello-world2.html` is like the following example:

HTML

```
<!DOCTYPE html>
<html>
<head>
    <script src="sdk/scripts/VSS.SDK.min.js"></script>
    <script type="text/javascript">
        VSS.init({
            explicitNotifyLoaded: true,
            usePlatformStyles: true
        });

        VSS.require(["AzureDevOps/Dashboards/WidgetHelpers",
"AzureDevOps/WorkItemTracking/RestClient"],
            function (WidgetHelpers, TFS_Wit_WebApi) {
                WidgetHelpers.IncludeWidgetStyles();
                VSS.register("HelloWorldWidget2", function () {
                    var projectId = VSS.getWebContext().project.id;

                    var getQueryInfo = function (widgetSettings) {
                        // Get a WIT client to make REST calls to Azure
DevOps Services
                        return
TFS_Wit_WebApi.getClient().getQuery(projectId, "Shared Queries/Feedback")
                            .then(function (query) {
                                // Create a list with query details
                                var $list = $('<ul>');
                                $list.append($('<li>').text("Query ID: " +
query.id));
```

```
                                $list.append($('<li>').text("Query Name: " +
query.name));
                                $list.append($('<li>').text("Created By: " +
(query.createdBy ? query.createdBy.displayName : "<unknown>")));

                                // Append the list to the query-info-
container
                                var $container = $('#query-info-container');
                                $container.empty();
                                $container.append($list);

                                // Use the widget helper and return success
as Widget Status
                                return
WidgetHelpers.WidgetStatusHelper.Success();
                            }, function (error) {
                                // Use the widget helper and return failure
as Widget Status
                                return
WidgetHelpers.WidgetStatusHelper.Failure(error.message);
                            });
                    }

                    return {
                        load: function (widgetSettings) {
                            // Set your title
                            var $title = $('h2.title');
                            $title.text('Hello World');

                            return getQueryInfo(widgetSettings);
                        }
                    }
                });
            VSS.notifyLoadSucceeded();
        });
    </script>

</head>
<body>
    <div class="widget">
        <h2 class="title"></h2>
        <div id="query-info-container"></div>
    </div>
</body>
</html>
```

Step 5: Update extension manifest

In this step, we update the extension manifest to include an entry for our second widget. Add a new contribution to the array in the `contributions` property and add the new file

`hello-world2.html` to the array in the files property. You need another preview image for the second widget. Name this `preview2.png` and place it in the `img` folder.

JSON

```
{
    ...,
    "contributions": [
        ...,
        {
            "id": "HelloWorldWidget2",
            "type": "ms.vss-dashboards-web.widget",
            "targets": [
                "ms.vss-dashboards-web.widget-catalog"
            ],
            "properties": {
                "name": "Hello World Widget 2 (with API)",
                "description": "My second widget",
                "previewImageUrl": "img/preview2.png",
                "uri": "hello-world2.html",
                "supportedSizes": [
                    {
                        "rowSpan": 1,
                        "columnSpan": 2
                    }
                ],
                "supportedScopes": ["project_team"]
            }
        }
    ],
    "files": [
        {
            "path": "hello-world.html",
            "addressable": true
        },
        {
            "path": "hello-world2.html",
            "addressable": true
        },
        {
            "path": "sdk/scripts",
            "addressable": true
        },
        {
            "path": "img",
            "addressable": true
        }
    ],
    "scopes": [
        "vso.work"
    ]
}
```

Step 6: Package, publish, and share

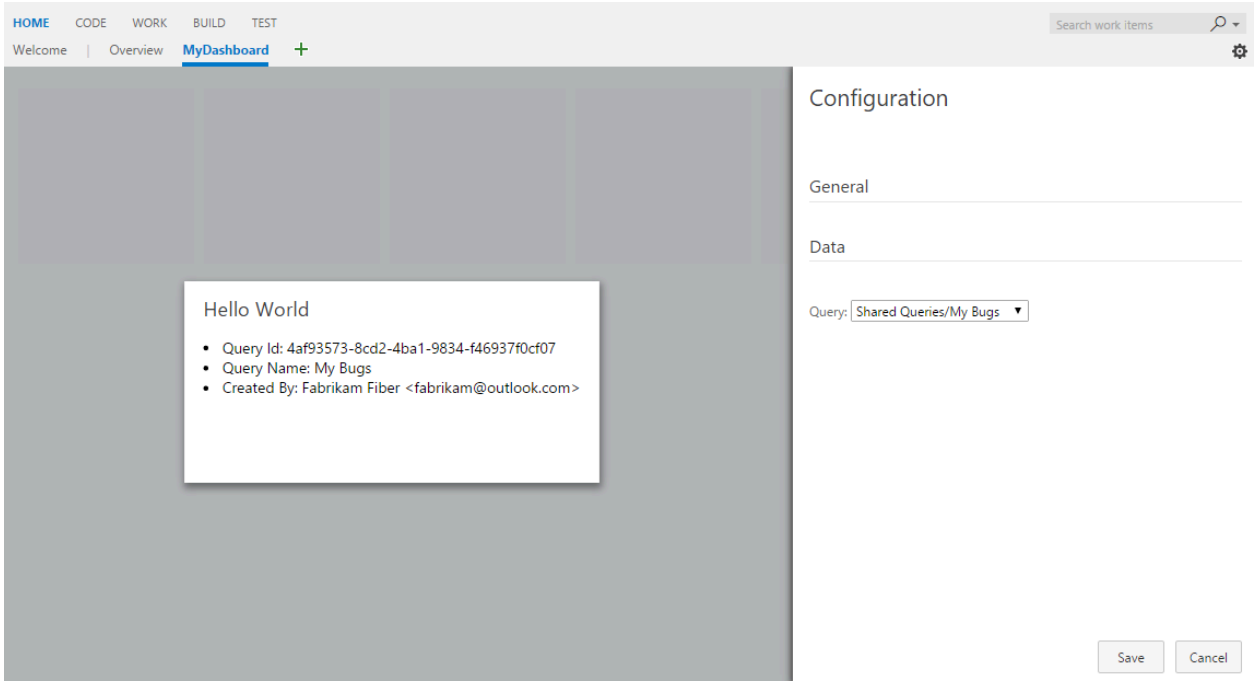
Package, publish, and share your extension. If you already published the extension, you can repackage the extension and directly update it to the Marketplace.

Step 7: Add widget from the catalog

Now, go to your team dashboard at `https:\//dev.azure.com/{Your_Organization}/{Your_Project}`. If this page is already open, then refresh it. Hover on **Edit** and select **Add**. The widget catalog opens where you find the widget you installed. To add it to your dashboard, choose your widget and select **Add**.

Part 3: Configure Hello World

In Part 2 of this guide, you saw how to create a widget that shows query information for a hard-coded query. In this part, we add the ability to configure the query to be used instead of the hard-coded one. When in configuration mode, the user gets to see a live preview of the widget based on their changes. These changes get saved to the widget on the dashboard when the user selects **Save**.



Step 1: Add HTML file

Implementations of Widgets and Widget Configurations are a lot alike. Both are implemented in the extension framework as contributions. Both use the same SDK file, `VSS.SDK.min.js`. Both are based on HTML, JavaScript, and CSS.

Copy the file `html-world2.html` from the previous example and rename the copy to `hello-world3.html`. Add another HTML file called `configuration.html`. Your folder now looks like the following example:

no-highlight

```
|--- README.md
|--- sdk
    |--- node_modules
    |--- scripts
        |--- VSS.SDK.min.js
|--- img
    |--- logo.png
|--- scripts
|--- configuration.html
|--- hello-world.html               // html page to be used for your widget
|--- hello-world2.html              // renamed copy of hello-world.html
|--- hello-world3.html              // renamed copy of hello-world2.html
|--- vss-extension.json             // extension's manifest
```

Add the following HTML in `configuration.html`. We basically add the mandatory reference to the `VSS.SDK.min.js` file and a `select` element for the dropdown to select a query from a preset list.

HTML

```
    <!DOCTYPE html>
    <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
            <script src="sdk/scripts/VSS.SDK.min.js"></script>
        </head>
        <body>
            <div class="container">
                <fieldset>
                    <label class="label">Query: </label>
                    <select id="query-path-dropdown" style="margin-
top:10px">
                        <option value="" selected disabled hidden>Please
select a query</option>
                        <option value="Shared Queries/Feedback">Shared
Queries/Feedback</option>
                        <option value="Shared Queries/My Bugs">Shared
Queries/My Bugs</option>
                        <option value="Shared Queries/My Tasks">Shared
Queries/My Tasks</option>
                    </select>
                </fieldset>
            </div>
        </body>
    </html>
```

Step 2: Configure JavaScript

Use JavaScript to render content in the widget configuration just like we did for the widget in Step 3 of Part 1 in this guide. This JavaScript code renders content, initializes the VSS SDK, maps the code for your widget configuration to the configuration name, and passes the configuration settings to the framework. In our case, the following code loads the widget configuration. Open the file `configuration.html` and the following `<script>` element to the `<head>`.

HTML

```
<script type="text/javascript">
    VSS.init({
        explicitNotifyLoaded: true,
        usePlatformStyles: true
    });

    VSS.require(["AzureDevOps/Dashboards/WidgetHelpers"], function
(WidgetHelpers) {
        VSS.register("HelloWorldWidget.Configuration", function () {
            var $queryDropdown = $("#query-path-dropdown");

            return {
                load: function (widgetSettings, widgetConfigurationContext)
{
                    var settings =
JSON.parse(widgetSettings.customSettings.data);
                    if (settings && settings.queryPath) {
                         $queryDropdown.val(settings.queryPath);
                     }

                    return WidgetHelpers.WidgetStatusHelper.Success();
                },
                onSave: function() {
                    var customSettings = {
                        data: JSON.stringify({
                                queryPath: $queryDropdown.val()
                            })
                    };
                    return
WidgetHelpers.WidgetConfigurationSave.Valid(customSettings);
                }
            }
        });
        VSS.notifyLoadSucceeded();
    });
</script>
```

- `VSS.init`, `VSS.require`, and `VSS.register` play the same role they played for the widget as described in Part 1. The only difference is that for widget configurations,

the function that is passed to `VSS.register` should return an object that satisfies the `IWidgetConfiguration` contract.

- The `load` property of the `IWidgetConfiguration` contract should have a function as its value. This function has the set of steps to render the widget configuration. In our case, it's to update the selected value of the dropdown element with existing settings if any. This function gets called when the framework instantiates your `widget configuration`
- The `onSave` property of the `IWidgetConfiguration` contract should have a function as its value. This function gets called by the framework when the user selects **Save** in the configuration pane. If the user input is ready to save, then serialize it to a string, form the `custom settings` object and use `WidgetConfigurationSave.Valid()` to save the user input.

In this guide, we use JSON to serialize the user input into a string. You can choose any other way to serialize the user input to string. It's accessible to the widget via the customSettings property of the `WidgetSettings` object. The widget has to deserialize, which is covered in Step 4.

Step 3: JavaScript - enable live preview

To enable live preview update when the user selects a query from the dropdown, we attach a change event handler to the button. This handler notifies the framework that the configuration changed. It also passes the `customSettings` to be used for updating the preview. To notify the framework, the `notify` method on the `widgetConfigurationContext` needs to be called. It takes two parameters, the name of the event, which in this case is `WidgetHelpers.WidgetEvent.ConfigurationChange`, and an `EventArgs` object for the event, created from the `customSettings` with the help of `WidgetEvent.Args` helper method.

Add the following code in the function assigned to the `load` property.

JavaScript

```
$queryDropdown.on("change", function () {
    var customSettings = {
       data: JSON.stringify({
               queryPath: $queryDropdown.val()
           })
    };
    var eventName = WidgetHelpers.WidgetEvent.ConfigurationChange;
    var eventArgs = WidgetHelpers.WidgetEvent.Args(customSettings);
```

```
    widgetConfigurationContext.notify(eventName, eventArgs);
});
```

Revised: Ensure that the framework is notified of the configuration change at least once to enable the **Save** button.

At the end, your `configuration.html` looks like the following example:

HTML

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
    <head>
        <script src="sdk/scripts/VSS.SDK.min.js"></script>
        <script type="text/javascript">
            VSS.init({
                explicitNotifyLoaded: true,
                usePlatformStyles: true
            });

            VSS.require(["AzureDevOps/Dashboards/WidgetHelpers"], function
(WidgetHelpers) {
                VSS.register("HelloWorldWidget.Configuration", function () {
                    var $queryDropdown = $("#query-path-dropdown");

                    return {
                        load: function (widgetSettings,
widgetConfigurationContext) {
                            var settings =
JSON.parse(widgetSettings.customSettings.data);
                            if (settings && settings.queryPath) {
                                $queryDropdown.val(settings.queryPath);
                             }

                             $queryDropdown.on("change", function () {
                                var customSettings = {data:
JSON.stringify({queryPath: $queryDropdown.val()})};
                                var eventName =
WidgetHelpers.WidgetEvent.ConfigurationChange;
                                var eventArgs =
WidgetHelpers.WidgetEvent.Args(customSettings);

widgetConfigurationContext.notify(eventName, eventArgs);
                             });

                            return
WidgetHelpers.WidgetStatusHelper.Success();
                        },
                        onSave: function() {
                            var customSettings = {data:
JSON.stringify({queryPath: $queryDropdown.val()})};
                            return
WidgetHelpers.WidgetConfigurationSave.Valid(customSettings);
```

```
                        }
                    }
                });
                VSS.notifyLoadSucceeded();
            });
        </script>
    </head>
    <body>
        <div class="container">
            <fieldset>
                <label class="label">Query: </label>
                <select id="query-path-dropdown" style="margin-top:10px">
                    <option value="" selected disabled hidden>Please select
a query</option>
                    <option value="Shared Queries/Feedback">Shared
Queries/Feedback</option>
                    <option value="Shared Queries/My Bugs">Shared Queries/My
Bugs</option>
                    <option value="Shared Queries/My Tasks">Shared
Queries/My Tasks</option>
                </select>
            </fieldset>
        </div>
    </body>
</html>
```

Step 4: Implement reload in the widget - JavaScript

We set up widget configuration to store the query path selected by the user. We now have to update the code in the widget to use this stored configuration instead of the hard-coded `Shared Queries/Feedback` from the previous example.

Open the file `hello-world3.html` and update the name of the widget from `HelloWorldWidget2` to `HelloWorldWidget3` in the line where you call `VSS.register`. This action allows the framework to uniquely identify the widget within the extension.

The function mapped to `HelloWorldWidget3` via `VSS.register` currently returns an object that satisfies the `IWidget` contract. Since our widget now needs configuration, this function needs to be updated to return an object that satisfies the `IConfigurableWidget` contract. To do so, update the return statement to include a property called reload per the following code. The value for this property is a function that calls the `getQueryInfo` method one more time. This reload method gets called by the framework every time the user input changes to show the live preview. This reload method is also called when the configuration is saved.

JavaScript

```
return {
    load: function (widgetSettings) {
        // Set your title
        var $title = $('h2.title');
        $title.text('Hello World');

        return getQueryInfo(widgetSettings);
    },
    reload: function (widgetSettings) {
        return getQueryInfo(widgetSettings);
    }
}
```

The hard-coded query path in `getQueryInfo` should be replaced with the configured query path, which can be extracted from the parameter `widgetSettings` that is passed to the method. Add the following code in the very beginning of the `getQueryInfo` method and replace the hard-coded query path with `settings.queryPath`.

JavaScript

```
var settings = JSON.parse(widgetSettings.customSettings.data);
if (!settings || !settings.queryPath) {
    var $container = $('#query-info-container');
    $container.empty();
    $container.text("Sorry nothing to show, please configure a query
path.");

    return WidgetHelpers.WidgetStatusHelper.Success();
}
```

At this point, your widget is ready to render with the configured settings.

Both the `load` and the `reload` properties have a similar function. This is the case for most simple widgets. For complex widgets, there would be certain operations that you would want to run just once no matter how many times the configuration changes. Or there might be some heavy-weight operations that need not run more than once. Such operations would be part of the function corresponding to the `load` property and not the `reload` property.

Step 5: Update extension manifest

Open the `vss-extension.json` file to include two new entries to the array in the `contributions` property. One for the `HelloWorldWidget3` widget and the other for its configuration. You need yet another preview image for the third widget. Name this

`preview3.png` and place it in the `img` folder. Update the array in the `files` property to include the two new HTML files we added in this example.

JSON

```
{
    ...
    "contributions": [
        ... ,
        {
             "id": "HelloWorldWidget3",
             "type": "ms.vss-dashboards-web.widget",
             "targets": [
                 "ms.vss-dashboards-web.widget-catalog",
                 "fabrikam.azuredevops-extensions-
myExtensions.HelloWorldWidget.Configuration"
             ],
             "properties": {
                 "name": "Hello World Widget 3 (with config)",
                 "description": "My third widget",
                 "previewImageUrl": "img/preview3.png",
                 "uri": "hello-world3.html",
                 "supportedSizes": [
                      {
                             "rowSpan": 1,
                             "columnSpan": 2
                         }
                     ],
                 "supportedScopes": ["project_team"]
             }
         },
         {
             "id": "HelloWorldWidget.Configuration",
             "type": "ms.vss-dashboards-web.widget-configuration",
             "targets": [ "ms.vss-dashboards-web.widget-configuration" ],
             "properties": {
                 "name": "HelloWorldWidget Configuration",
                 "description": "Configures HelloWorldWidget",
                 "uri": "configuration.html"
             }
         }
    ],
    "files": [
            {
                "path": "hello-world.html", "addressable": true
            },
             {
                "path": "hello-world2.html", "addressable": true
            },
            {
                "path": "hello-world3.html", "addressable": true
            },
            {
                "path": "configuration.html", "addressable": true
```

```
            },
            {
                "path": "sdk/scripts", "addressable": true
            },
            {
                "path": "img", "addressable": true
            }
        ],
        ...
}
```

The contribution for widget configuration follows a slightly different model than the widget itself. A contribution entry for widget configuration has:

- The **ID** to identify your contribution. The ID should be unique within an extension.
- The **type** of contribution. For all widget configurations, it should be `ms.vss-dashboards-web.widget-configuration`
- The array of **targets** to which the contribution is contributing. For all widget configurations, it has a single entry: `ms.vss-dashboards-web.widget-configuration`.
- The **properties** that contain a set of properties that includes name, description, and the URI of the HTML file used for configuration.

To support configuration, the widget contribution needs to be changed as well. The array of **targets** for the widget needs to be updated to include the ID for the configuration in the form <`publisher`>.<`id for the extension`>.<`id for the configuration contribution`> which in this case is `fabrikam.vsts-extensions-myExtensions.HelloWorldWidget.Configuration`.

> ⚠ **Warning**
>
> If the contribution entry for your configurable widget doesn't target the configuration using the right publisher and extension name as described previously, the configure button doesn't show up for the widget.

At the end of this part, the manifest file should contain three widgets and one configuration. You can get the complete manifest from the sample here.

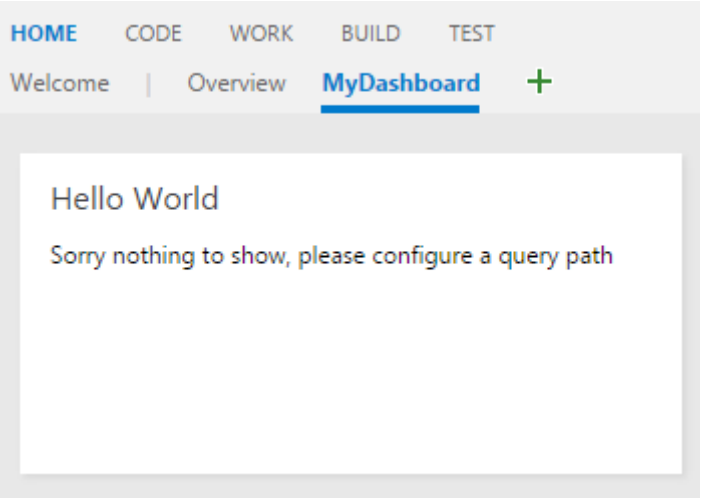
Step 6: Package, publish, and share

If your extension isn't published, see this section. If you already published the extension, you can repackage the extension and directly update it to the Marketplace.

Step 7: Add widget from the catalog

Now, go to your team dashboard at https://dev.azure.com/{Your_Organization}/{Your_Project}. If this page is already open, refresh it. Hover on **Edit** and select **Add**. This action should open the widget catalog where you find the widget you installed. To add the widget to your dashboard, choose your widget and select **Add**.

A message similar to the following, asks you to configure the widget.



There are two ways to configure widgets. One is to hover on the widget, select the ellipsis that appears on the top-right corner and then select Configure. The other is to select the Edit button in the bottom right of the dashboard, and then select the configure button that appears on the top-right corner of the widget. Either opens the configuration experience on the right side, and a preview of your widget in the center. Go ahead and choose a query from the dropdown. The live preview shows the updated results. Select **Save** and your widget displays the updated results.

Step 8: Configure more (optional)

You can add as many HTML form elements as you need in the `configuration.html` for more configuration. There are two configurable features available out-of-the-box: Widget name and widget size.

By default, the name that you provide for your widget in the extension manifest is stored as the widget name for every instance of your widget that ever gets added to a dashboard. You can allow users to configure, so that they can add any name they want to their instance of your widget. To allow such configuration, add `isNameConfigurable:true` in the properties section for your widget in the extension manifest.

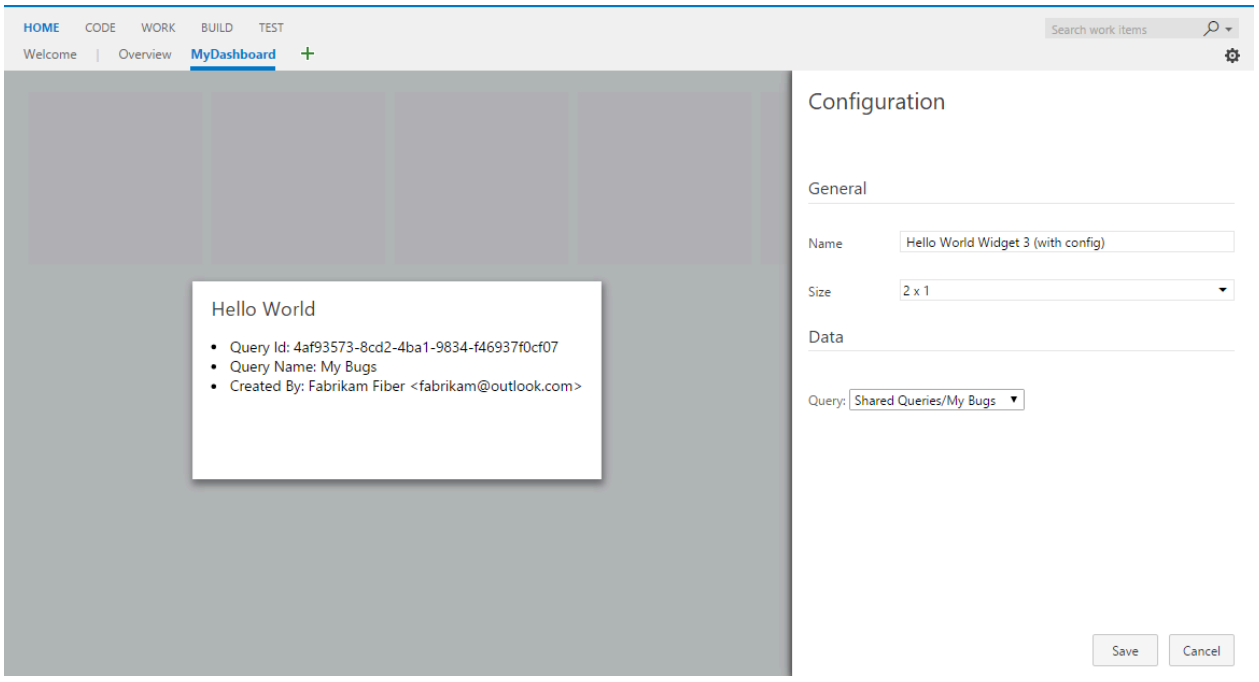
If you provide more than one entry for your widget in the `supportedSizes` array in the extension manifest, then users can configure the widget's size as well.

The extension manifest for the third sample in this guide would look like the following example if we enable the widget name and size configuration:

JSON

```
{
    ...
    "contributions": [
        ... ,
        {
             "id": "HelloWorldWidget3",
             "type": "ms.vss-dashboards-web.widget",
             "targets": [
                 "ms.vss-dashboards-web.widget-catalog",
                 "fabrikam.azuredevops-extensions-
myExtensions.HelloWorldWidget.Configuration"
             ],
             "properties": {
                 "name": "Hello World Widget 3 (with config)",
                 "description": "My third widget",
                 "previewImageUrl": "img/preview3.png",
                 "uri": "hello-world3.html",
                 "isNameConfigurable": true,
                 "supportedSizes": [
                    {
                        "rowSpan": 1,
                        "columnSpan": 2
                    },
                    {
                        "rowSpan": 2,
                        "columnSpan": 2
                    }
                 ],
                 "supportedScopes": ["project_team"]
             }
         },
         ...
    ]
}
```

With the previous change, repackage and update your extension. Refresh the dashboard that has this widget (Hello World Widget 3 (with config)). Open the configuration mode for your widget, you should now be able to see the option to change the widget name and size.



Choose a different size from the drop-down. You see the live preview get resized. Save the change and the widget on the dashboard is resized as well.

Changing the name of the widget doesn't result in any visible change in the widget because our sample widgets don't display the widget name anywhere. Let us modify the sample code to display the widget name instead of the hard-coded text "Hello World."

To do so, replace the hard-coded text "Hello World" with `widgetSettings.name` in the line where we set the text of the `h2` element. This action ensures that the widget name gets displayed every time the widget gets loaded on page refresh. Since we want the live preview to be updated every time the configuration changes, we should add the same code in the `reload` part of our code as well. The final return statement in `hello-world3.html` is as follows:

JavaScript

```
return {
    load: function (widgetSettings) {
        // Set your title
        var $title = $('h2.title');
        $title.text(widgetSettings.name);

        return getQueryInfo(widgetSettings);
    },
    reload: function (widgetSettings) {
        // Set your title
        var $title = $('h2.title');
        $title.text(widgetSettings.name);

        return getQueryInfo(widgetSettings);
```

```
    }
}
```

Repackage and update your extension again. Refresh the dashboard that has this widget.

Any changes to the widget name, in the configuration mode, update the widget title now.

Feedback

Was this page helpful? Yes No

Provide product feedback

Add a chart

Article • 03/13/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Learn how you can add charts to your extensions. Charts can be added to any Azure DevOps Services extension.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Charts are easy to create, resizable, interactive, and consistent with the Azure DevOps Services look and feel. The following chart types are supported:

- Line
- Bar
- Column
- Area
- Stacked bar
- Stacked column
- Stacked area
- Pie
- Pivot table
- Histogram

If you're in a hurry and want to get your hands on the code right away, you can download the complete samples. Once downloaded, go to the `charts` folder, and then follow the packaging and publishing instructions to publish the sample extension. The extension contains sample chart widgets.

How to organize your code

For this tutorial, we create a widget and add a chart to it. To do so, in the `home` folder for your project, create a `chart.html` file with the following contents:

HTML file

```
HTML
```

```
<!DOCTYPE html>
<html>
<head>
    <script src="sdk/scripts/VSS.SDK.js"></script>
    <script src="scripts/pie-chart.js"></script>
</head>
<body>
    <div class="widget">
        <h2 class="title">Chart Widget</h2>
        <div id="Chart-Container"></div>
    </div>
</body>
</html>
```

> Use the **npm install** command to retrieve the SDK: `npm install vss-web-extension-sdk`. To learn more about the SDK, visit the Client SDK GitHub Page.

Ensure that the `VSS.SDK.js` file is inside the `sdk/scripts` folder so that the path is `home/sdk/scripts/VSS.SDK.js`.

Images

Add images to an `img` folder in your project directory so that the path is `home/img/logo.png` and `home/img/CatalogIcon.png`. See our newest documentation, the Formula Design System and Azure DevOps extension sample SDK.

Extension manifest file

In the `home` folder of your project, create your extension manifest file. Create a `vss-extension.json` file with the following contents:

JSON

```
{
    "manifestVersion": 1,
    "id": "samples-charts",
    "version": "1.0.0",
    "name": "Interactive Charts in Azure DevOps Services",
    "description": "Samples of interactive charts from the Chart SDK.",
    "publisher": "Fabrikam",
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "icons": {
```

```
        "default": "img/logo.png"
    },
    "demands": ["contribution/ms.vss-dashboards-web.widget-sdk-version-2",
"contribution/ms.vss-web.charts-service"],
    "contributions": [
        {
            "id": "chart",
            "type": "ms.vss-dashboards-web.widget",
            "targets": [
                "ms.vss-dashboards-web.widget-catalog"
            ],
            "properties": {
                "name": "Sample Chart",
                "description": "A simple chart widget with no configuration
and hard-coded data.",
                "catalogIconUrl": "img/CatalogIcon.png",
                "uri": "chart.html",
                "supportedSizes": [
                    {
                        "rowSpan": 2,
                        "columnSpan": 2
                    }
                ],
                "supportedScopes": [
                    "project_team"
                ]
            }
        }
    ],
    "files": [
        {
            "path": "chart.html",
            "addressable": true
        },
        {
            "path": "sdk/scripts/VSS.SDK.js",
            "addressable": true
        },
        {
            "path": "img",
            "addressable": true
        },
        {
            "path": "scripts",
            "addressable": true
        }
    ],
    "scopes": [
    ]
}
```

Before uploading this extension, you must update the `publisher` to yours.

Put the following code snippets into a `chart.js` file in a `scripts` folder, so that the path is `home/scripts/chart.js`. Then, follow the packaging and publishing instructions to publish your extension.

Charts

Chart

This sample renders a chart. The `data` and `labelValues` have been hardcoded here, and would need to be changed to the data you want to visualize.

JavaScript

```
VSS.init({
        explicitNotifyLoaded: true,
        usePlatformStyles: true
    });

VSS.require([
        "TFS/Dashboards/WidgetHelpers",
        "Charts/Services"
        ],
        function (WidgetHelpers, Services) {
        WidgetHelpers.IncludeWidgetStyles();
        VSS.register("chart", function () {
             return {
             load:function() {
                return
Services.ChartsService.getService().then(function(chartService){
                    var $container = $('#Chart-Container');
                    var chartOptions = {
                        "hostOptions": {
                            "height": "290",
                            "width": "300"
                        },
                        "chartType": "pie",
                        "series": [{
                            "data": [11, 4, 3, 1]
                        }],
                        "xAxis": {
                            "labelValues": ["Design", "On Deck",
"Completed", "Development"]
                        },
                        "specializedOptions": {
                            "showLabels": "true",
                            "size": 200
                        }
                    };
```

```
                    chartService.createChart($container, chartOptions);
                    return WidgetHelpers.WidgetStatusHelper.Success();
                });
                }
            }
        });
    VSS.notifyLoadSucceeded();
});
```

Here, the chart's size is defined in `hostOptions`. The series property is an array and contains a single object with data in it. The `xAxis` object contains `labelValues`, which correspond to the `data`. For pie charts, we also have some special options that are defined by the `specializedOptions` property. Here, we're explicitly enabling data labels for the pie chart. We also need to set the size of the pie chart by specifying its outer diameter.

The chart requires a container, the chart options, and a call to the Chart Service to initialize and render. For more information on chart options, see vss-web-extension-sdk/typings/charts.

Stacked area chart

The following sample renders a stacked area chart. This chart type is ideal to compare a relationship of parts to a whole and highlight general trends across categories. It's commonly used to compare trends over time. This sample also specifies a custom color for one of the series being rendered.

JavaScript

```
VSS.init({
        explicitNotifyLoaded: true,
        usePlatformStyles: true
    });

VSS.require([
        "TFS/Dashboards/WidgetHelpers",
        "Charts/Services"
        ],
        function (WidgetHelpers, Services) {
        WidgetHelpers.IncludeWidgetStyles();
        VSS.register("StackedAreaChart", function () {
             return {
             load:function() {
                return
Services.ChartsService.getService().then(function(chartService){
                    var $container = $('#Chart-Container');
                    var chartOptions = {
                        "hostOptions": {
```

```
                        "height": "290",
                        "width": "300"
                    },
                    "chartType": "stackedArea",
                    "series": [
                        {
                            "name": "Completed",
                            "data": [1,3,4,3,6,1,9,0,8,11]
                        },
                        {
                            "name": "Development",
                            "data": [1,1,0,3,0,2,8,9,2,8]
                        },
                        {
                            "name": "Design",
                            "data": [0,0,0,6,1,1,9,9,1,9],
                            "color": "#207752"
                        },
                        {
                            "name": "On Deck",
                            "data": [1,2,4,5,4,2,1,7,0,6]
                        }
                    ],
                    "xAxis": {
                        "labelFormatMode": "dateTime_DayInMonth",
                        "labelValues": [
                            "1/1/2016",
                            "1/2/2016",
                            "1/3/2016",
                            "1/4/2016",
                            "1/5/2016",
                            "1/6/2016",
                            "1/7/2016",
                            "1/8/2016",
                            "1/9/2016",
                            "1/10/2016"
                        ]
                    }
                }
                chartService.createChart($container, chartOptions);
                return WidgetHelpers.WidgetStatusHelper.Success();
            });
            }
        }
        });
    VSS.notifyLoadSucceeded();
});
```

Pivot table

This sample renders a Pivot Table. This chart type helps arrange and summarize complex data in a tabular form.

JavaScript

```
VSS.init({
        explicitNotifyLoaded: true,
        usePlatformStyles: true
    });

VSS.require([
        "TFS/Dashboards/WidgetHelpers",
        "Charts/Services"
        ],
        function (WidgetHelpers, Services) {
        WidgetHelpers.IncludeWidgetStyles();
        VSS.register("PivotTable", function () {
             return {
             load:function() {
                return
Services.ChartsService.getService().then(function(chartService){
                    var $container = $('#Chart-Container');
                    var chartOptions = {
                        "hostOptions": {
                            "height": "290",
                            "width": "300"
                        },
                        "chartType": "table",
                        "xAxis": {
                            "labelValues": ["Design","In
Progress","Resolved","Total"]
                        },
                        "yAxis": {
                            "labelValues": ["P1","P2","P3","Total"]
                        },
                        "series": [
                            {
                                "name": "Design",
                                "data": [
                                    [0,0,1],
                                    [0,1,2],
                                    [0,2,3]
                                ]
                            },
                            {
                                "name": "In Progress",
                                "data": [
                                    [1,0,4],
                                    [1,1,5],
                                    [1,2,6]
                                ]
                            },
                            {
                                "name": "Resolved",
                                "data": [
                                    [2,0,7],
                                    [2,1,8],
```

```
                                [2,2,9]
                            ]
                        },
                        {
                            "name": "Total",
                            "data": [
                                [3,0,12],
                                [3,1,15],
                                [3,2,18],
                                [0,3,6],
                                [1,3,15],
                                [2,3,24],
                                [3,3,10]
                            ],
                            "color": "rgba(255,255,255,0)"
                        }
                    ]
                }

                    chartService.createChart($container, chartOptions);
                    return WidgetHelpers.WidgetStatusHelper.Success();
                });
                }
            }
        });
    VSS.notifyLoadSucceeded();
});
```

Call a REST API

Article • 05/30/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

To call a REST API from your extension, get the client service first, and then use that to call the API.

> **Note**
>
> Looking for Azure DevOps REST APIs? See the latest **Azure DevOps REST API reference**.
>
> For information about .NET client libraries, see **.NET client libraries for Azure DevOps**.

Client services

You can find the full list of available REST clients and their references here.

Call an API

> You'll have to add the scope `"scopes": ["vso.work"],` to your `vss-extension.json` to use the work item tracking client.

1. Get the REST client. In this case, we're getting the work item tracking client.

   JavaScript

   ```
   VSS.require(["VSS/Service", "TFS/WorkItemTracking/RestClient"],
   function (VSS_Service, TFS_Wit_WebApi) {
       // Get the REST client
       var witClient =
   VSS_Service.getCollectionClient(TFS_Wit_WebApi.WorkItemTrackingHttpClie
   nt);
       // ...
   });
   ```

2. Call the API, `getWorkItems`, using the client service, `witClient`, with a callback that handles results.

JavaScript

```
witClient.getWorkItems(/* some work item IDs */ [1,2,3,4],
["System.Title"]).then(
    function(workItems) {
        console.log(JSON.stringify(workItems));
    });
```

Next steps

Display the results of your REST API call in a Grid.

Public project support by Azure DevOps Services extensions

Article • 10/04/2022

Azure DevOps Services

Before public project support, all Azure DevOps Services projects were private. Private projects only authenticated users with access to the project, so the public couldn't see or interact with it. A public project allows non-member users to view the contents of that project in a read-only state.

> A non-member user is either **anonymous** (not authenticated to Azure DevOps Services) or **public** (are authenticated to Azure DevOps Services, but do not belong to the organization).

Non-member users generally see the same views as authenticated users, however non-public functionality is hidden or disabled. An example of non-public functionality, such as settings and actions, includes a queue build.

> **Note**
>
> Azure DevOps Services public project support is currently in **Limited Preview**. Contact **vsts-public@microsoft.com** if you are interested in developing extensions that support public projects. To learn more about public projects, see **Azure DevOps Services Public Projects Limited Preview**.

Decide whether to make an extension visible to non-member users

As the extension developer, you can make all or part of your extension available to non-member users. These users can only use your extension from within a public project. If you choose to not make your extension available to non-member users, no change is required and there's no impact on members, whether they use your extension from within a public project.

Use this checklist to help decide if you should make your extension available to non-member users:

- ✓ Data presented by your extension is relevant to non-member users

- ✔ Your extension contributes capabilities at the project level
- ✔ Your extension contributes to areas of the product that are accessible by non-member users
- ✔ Your extension doesn't extend or rely on features that are unavailable to non-member users, for example the Data Service or certain Azure DevOps Services REST APIs. For more information, see the Limitations section of this article.

Contribution visibility

By default, contributions are only visible to organization members. To give non-member users visibility to a contribution, set the `restrictedTo` attribute on that contribution. The value is a string array that lists which types of users should have visibility to the contribution. The possible values are:

- `member` an authenticated user that is a member of the organization
- `public` an authenticated user that is **not** a member of the organization
- `anonymous` an unauthenticated user

Example: make a hub visible to anonymous, public, and member users

JSON

```
{
    "contributions": [
        {
            "id": "my-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-code-web.code-hub-group"
            ],
            "restrictedTo": [
                "member",
                "public",
                "anonymous"
            ],
            "properties": {
                ...
            }
        }
    ]
}
```

You can also set the default visibility for all contributions in your extension by setting the `restrictedTo` attribute at the root your extension manifest. You can then override this

default on individual contributions.

Example: make every contribution, except for one, visible to all users

JSON

```
{
    "id:": "my-extension",
    "name": "My Extension",
    "version": "1.0.0",
    ...
    "restrictedTo": [
           "anonymous",
           "public",
           "member"
    ],
    "contributions": [
        {
            "id": "my-member-only-widget",
            "type": "ms.vss-dashboards-web.widget",
            "restrictedTo": [
                "member"
            ],
            "properties": {
                ...
            }
        },
        {
            "id": "my-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-code-web.code-hub-group"
            ],
            "properties": {
                ...
            }
        },
        {
            "id": "my-second-hub",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-work-web.work-hub-group"
            ],
            "properties": {
                ...
            }
        }
    ]
}
```

Limitations

If you want to make some or all aspects of your contribution available to public users, be aware of the following limitations.

VSS SDK methods

The core SDK script, VSS.SDK.js, allows web extensions to communicate with the parent frame to do operations like initializing communication and getting context information about the current user. The following VSS SDK methods aren't supported for non-member users:

- `VSS.getAccessToken()`
- `VSS.getAppToken()`

Extension data service

Because data managed by the extension data service isn't scoped or secured to a project, non-member users can't read or write any type of extension data.

Example: handling data access errors

If the data service can't access the data service because of permission limitations by the calling user, the promise returned from the call to `getValue` is rejected. The error passed to the reject function has a name, which can help you understand why the call failed to read or write data.

JavaScript

```
VSS.getService(VSS.ServiceIds.ExtensionData).then(function(dataService) {
    dataService.getValue("someKey").then(function(value) {
         // do something with the value
    }, function(error) {
       if (error.name === "AccessCheckException") {
           alert(error.message);
       }
    });
});
```

REST APIs

A limited set of Azure DevOps Services REST APIs is available to non-member users. These APIs include most organization-level and project-level APIs for features unavailable to non-member users in general. Consider this information when you're deciding whether to make your extension available to non-member users.

We recommend that you use version 5.0 and later APIs, as certain APIs are only available to non-member users starting with version 5.0.

Identity references

A majority of Azure DevOps Services REST APIs use a common "contract" to represent a user or group. To protect member information, like email addresses, when a REST API is invoked by an anonymous or public user, certain fields, like `uniqueName` are omitted.

Checking user permissions

Use permissions to decide whether to surface or enable a capability in your extension. The Security REST API gets used from web extension code to check whether the current user has permission in Azure DevOps Services to complete the task. This way, the user won't think they have permission to do something, only to find that they don't.

Example: check whether the user has permission to queue builds

This example shows how to use the Security REST client to check whether the user has permissions to queue builds in the current project. By default, non-member users don't have this permission.

JavaScript

```
VSS.require(["VSS/Service", "VSS/security/RestClient"],
function(VSS_Service, Security_RestClient) {
   var client =
VSS_Service.getCollectionClient(Security_RestClient.SecurityHttpClient3);

   var securityToken = VSS.getWebContext().project.id;

   client.hasPermissionsBatch({
    evaluations: [
       {
           "securityNamespaceId": "33344D9C-FC72-4d6f-ABA5-FA317101A7E9",
           "token": securityToken,
           "permissions": 128 /* queue builds */
       }
    ],
    alwaysAllowAdministrators: true
```

```
}
).then(function(response) {
     console.log("Can user queue builds in this project? " +
response.evaluations[0].value);
  });
});
```

Dashboard widget considerations

Just like other types of contributions, the visibility of dashboard widget contributions is controlled with the `restrictedTo` contribution property. For example to make a widget visible to both non-member and member users:

JSON

```
{
  "contributions": [
    {
      "id": "HelloWorldWidget",
      "type": "ms.vss-dashboards-web.widget",
      "targets": [
        "ms.vss-dashboards-web.widget-catalog"
      ],
      "restrictedTo": [
        "member",
        "public",
        "anonymous"
      ],
      "properties": {
          ...
      }
    }
  ]
}
```

Widget settings

While controlling visibility of the widget to non-member users, the dashboard framework also provides an optional, open-form storage mechanism for widget settings. Two mechanisms are available for indicate whether widget settings are available for use by non-member users in public projects.

A widget with configurable settings that is visible to non-member users **must** follow one of the following patterns. Not doing so blocks the widget from surfacing to these users.

Pattern 1: widget declares its instances only store project-specific settings

Set the widget contribution's `canStoreCrossProjectSettings` property to `false`, indicating the widget settings are project-specific.

JSON

```
{
    "id:": "HelloWorldWidget",
    "type": "ms.vss-dashboards-web.widget",
    ...
    "properties": {
        "canStoreCrossProjectSettings": false
    }
}
```

Pattern 2: a widget instance declares its settings are project-specific

Individual widget instances can also indicate that its settings are project-specific and available to non-member users. When saving the settings, the widget should set the `hasCrossProjectSettings` to `false` in the stringified JSON string:

JSON

```
{
    "hasCrossProjectSettings": false,
    "hypotheticalWidgetOption": true,
    "backlogLevel": "Stories"
}
```

Build and release considerations

If your extension contributes a build or release task, no change is required to use that task from a pipeline in a public project.

Work item tracking considerations

Extensions don't work for non-member users in the context of a public project without changes. This includes the work item form, other work item experiences, or interaction with work item tracking REST APIs.

Work item form

All work item updates or deletes fail for non-member users.

Identities

In Azure DevOps Services REST API version 5.0 and later, identities are returned as `IdentityRef` objects instead of strings. As described previously, certain fields, like `uniqueName` aren't returned in these objects if the API call was made by a non-member user.

APIs

Only project-scoped REST APIs can be invoked by an extension when the current user isn't an organization member. Any REST API calls not scoped to a project are rejected.

Queries

There are the following limitations for non-member users related to work item queries:

- non-member users can run known queries by ID or path
- Queries must be scoped to the current project. Any work items not belonging to the current project are excluded
- non-member user can't create new queries or execute WIQL queries

Package and publish extensions

Article • 01/09/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Once you develop your extension, you can package and publish it to the Visual Studio Marketplace. The Marketplace is a global repository for private and public extensions, integrations, and other offers from Microsoft.

> **Note**
>
> For information on the discovery properties available in your extension's manifest file that helps users discover and learn about your extension, see the Extension Manifest Reference.

Prerequisites

The following list of requirements must be met before you publish to the Marketplace.

- Install the extension packaging tool (TFX). Run `npm install -g tfx-cli` from a command prompt.
- Ensure the proper permissions are granted to use any images, for example, icons, logos, screenshots, and so on.
- Include a thorough `overview.md` file to describe your listing in the Marketplace.
- Include an icon for your extension, which is at least 128x128 pixels in size.
- When you refer to Microsoft products, use full names in place of abbreviations, for example, Azure DevOps vs. AzDO or - any other abbreviation.
- Refrain from using brand names in the name of your extension.

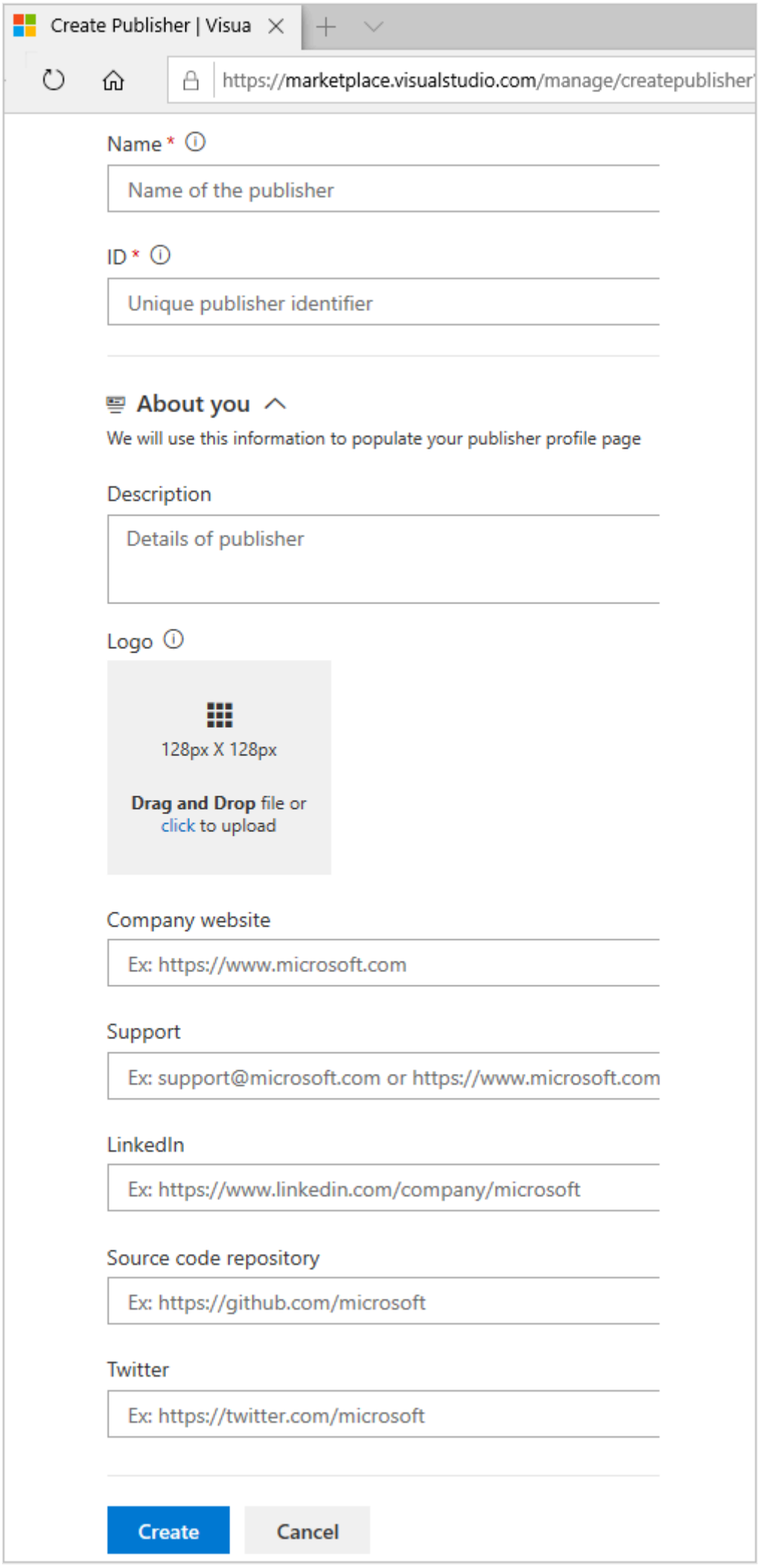
Create a publisher

All extensions and integrations, including extensions from Microsoft, have a publisher. Anyone can create a publisher and publish extensions under it. You can also give other people access to your publisher if a team is developing the extension.

A user owns the publisher, typically the user who created it. You can also share the publisher with other users.

1. Sign in to the Visual Studio Marketplace Publishing Portal.

2. If you're not already a member of an existing publisher, **+ Create a publisher**. Enter a name in the publisher name field. The ID field should automatically get set based on the name you entered.

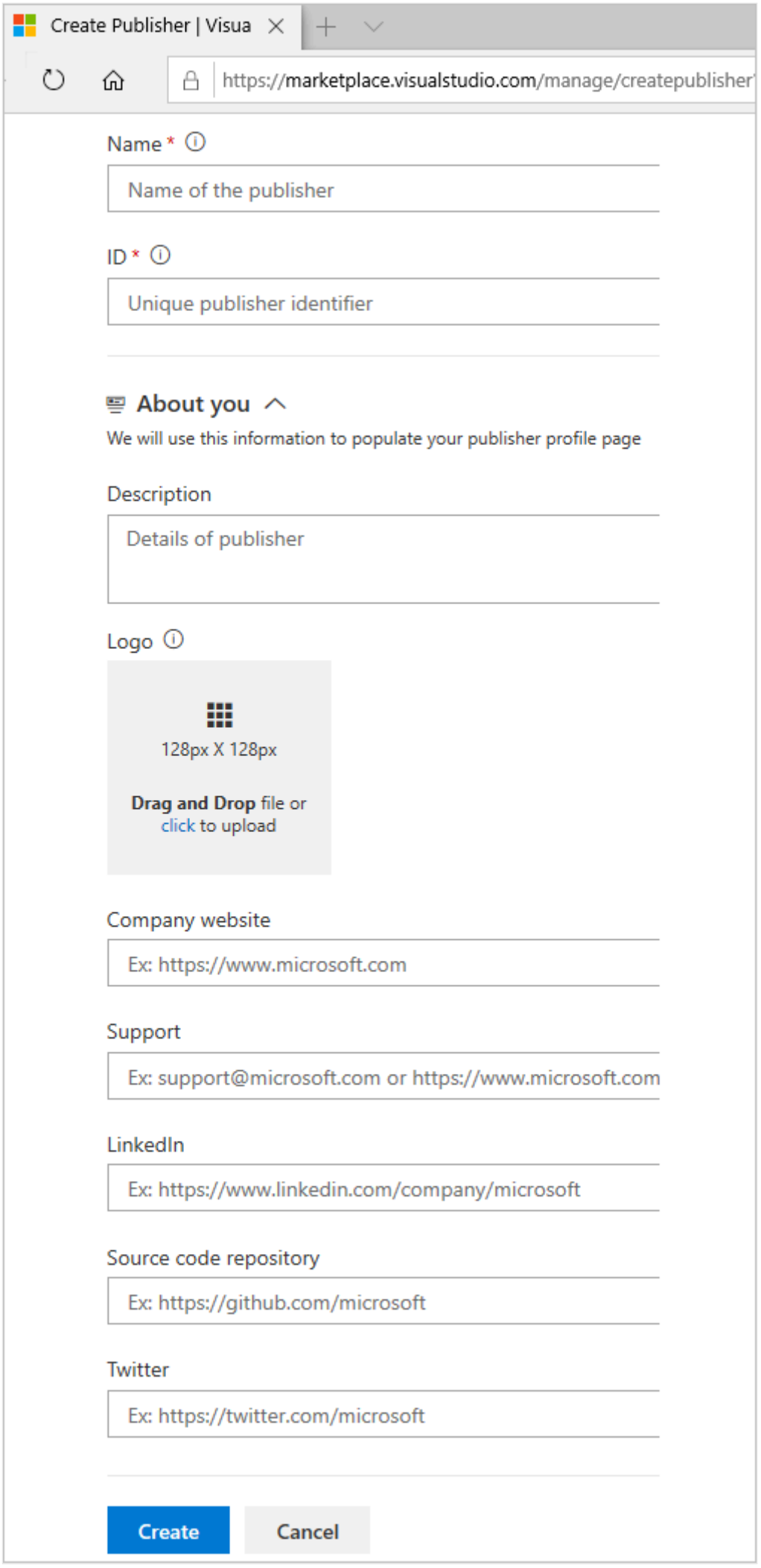

Create Publisher | Visua
https://marketplace.visualstudio.com/manage/createpublisher
Name *
Name of the publisher
ID *
Unique publisher identifier
About you
We will use this information to populate your publisher profile page
Description
Details of publisher
Logo
128px X 128px
Drag and Drop file or
click to upload
Company website
Ex: https://www.microsoft.com
Support
Ex: support@microsoft.com or https://www.microsoft.com
LinkedIn
Ex: https://www.linkedin.com/company/microsoft
Source code repository
Ex: https://github.com/microsoft
Twitter
Ex: https://twitter.com/microsoft
Create
Cancel

> **Note**
>
> Make note of the ID, as you need to set it in the manifest file of your extension.

If you're not prompted to create a publisher, scroll down to the bottom of the page and select **Publish extensions** below *Related sites*.

- Specify an identifier for your publisher, for example: `mycompany-myteam`. This identifier is used as the value for the `publisher` attribute in your extension manifest file.
- Specify a display name for your publisher, for example: `My Team`

3. Review the Marketplace Publisher Agreement, and then select **Create**.


Create Publisher | Visua
https://marketplace.visualstudio.com/manage/createpublisher
Name *
Name of the publisher
ID *
Unique publisher identifier
About you
We will use this information to populate your publisher profile page
Description
Details of publisher
Logo
128px X 128px
Drag and Drop file or
click to upload
Company website
Ex: https://www.microsoft.com
Support
Ex: support@microsoft.com or https://www.microsoft.com
LinkedIn
Ex: https://www.linkedin.com/company/microsoft
Source code repository
Ex: https://github.com/microsoft
Twitter
Ex: https://twitter.com/microsoft
Create
Cancel

Once the publisher's created, you're directed to manage items, but there aren't any items.

Package your extension

To upload your extension, you need to package it as a VSIX 2.0-compatible .vsix file. Microsoft provides a cross-platform command-line interface (CLI) to package and publish your extension.

1. Open your extension manifest file (`vss-extension.json`) and set the value of the `publisher` field to the ID of your publisher. For example:

   JSON

   ```
   {
       ...
       "id": "my-first-extension",
       "publisher": "AnnetteNielsen",
       ...
   }
   ```

2. From a command prompt, run the TFX tool's packaging command from your extension directory.

   ```
   npx tfx-cli extension create
   ```

   A message displays indicating your extension is successfully packaged:

   ```
   === Completed operation: create extension ===
   - VSIX: C:\my-first-extension\AnnetteNielsen.my-first-extension-
   1.0.0.vsix
   - Extension ID: my-first-extension
   - Extension Version: 1.0.0
   - Publisher: AnnetteNielsen
   ```

> **Note**
>
> An extension/integration's version must be incremented on every update.
> If you haven't incremented your extension/integration in the manifest, you should

pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and saves the new version to your manifest.

Check package size

Check the size of the vsix after it gets packaged. If it's greater than 50 MB, you need to optimize it. To do so, see the following considerations:

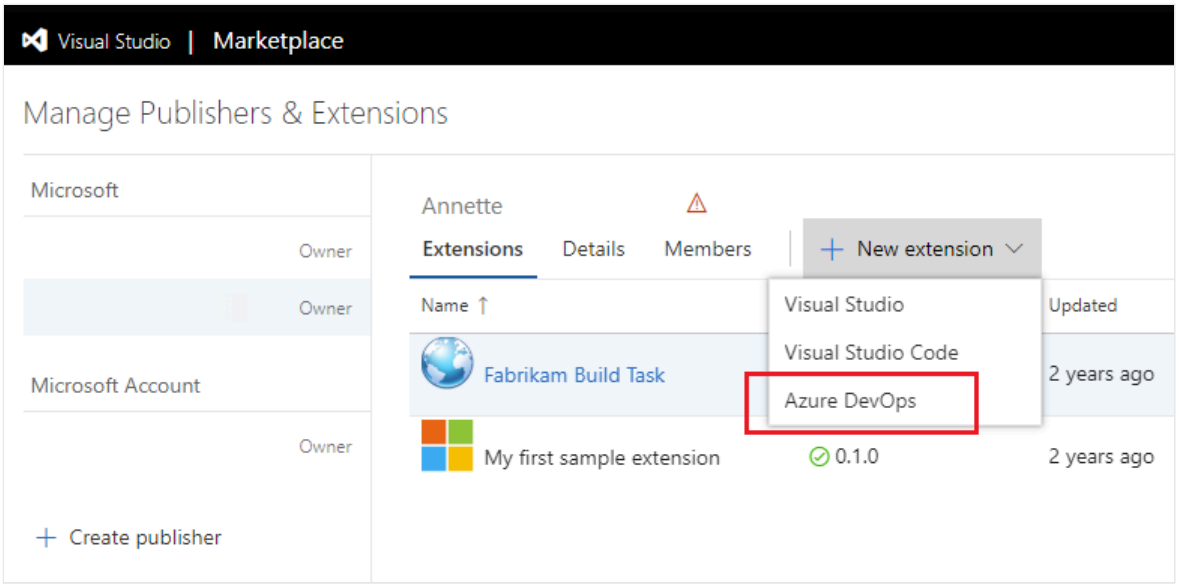
- Deduplicate the common dependencies, if any, by stating them once in the extension package.
- Fetch things at runtime or during install time rather than providing it within the package. Consider using the tool installer lib to pull tool dependencies at runtime. Using the lib offers benefits where the tool gets cached by version so for private agents, it doesn't download every build. We made it a lib so it can be used outside of tool installer tasks. But, the task doesn't work in disconnected scenarios (no internet), which should be in the description / docs for the task.
- Some users have success with WebPack to tree shake their dependencies in their tasks.

Publish your extension

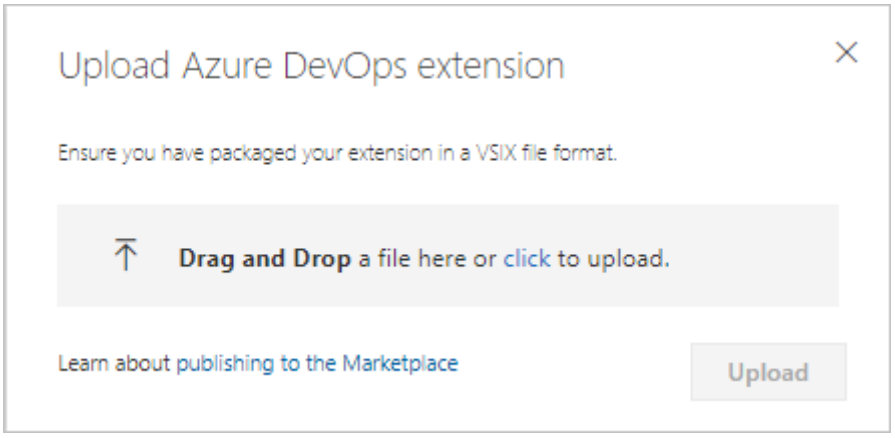
Once your extension is packaged, you can upload it to the Marketplace under a publisher. The `publisher` identifier specified in your extension's manifest file must match the identifier of the publisher the extension is uploaded under.

1. From the management portal, select your publisher from the drop-down menu at the top of the page.

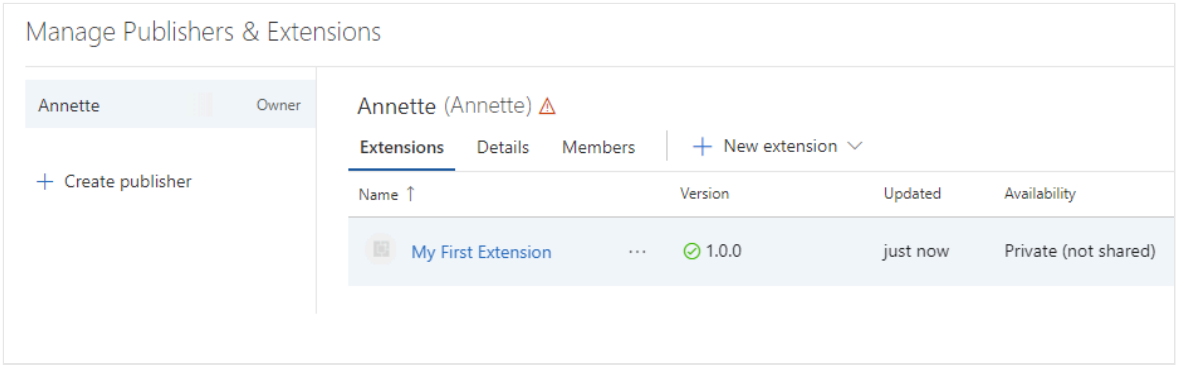
2. Select **New extension** > **Azure DevOps**.



3. Drag and drop your file or select it to find your VSIX file, which you created in the previous packaging step, and then choose **Upload**.



After quick validation, your extension appears in the list of published extensions. Don't worry, the extension is only visible to you.



At this point, your extension isn't visible to any accounts and can't be installed until you share it.

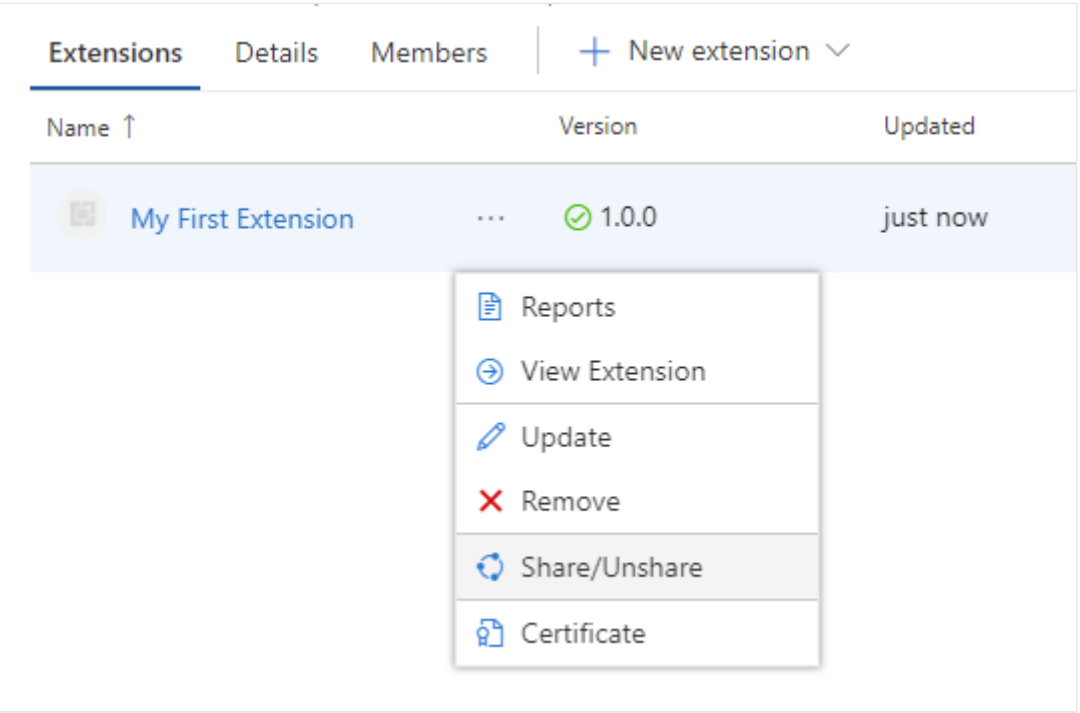
> **Note**

Microsoft runs a virus scan on each new and updated extension package published. Until the scan is all clear, we don't publish the extension in the Marketplace for public usage. This way we also avoid surfacing inappropriate or offensive content on the Marketplace pages.

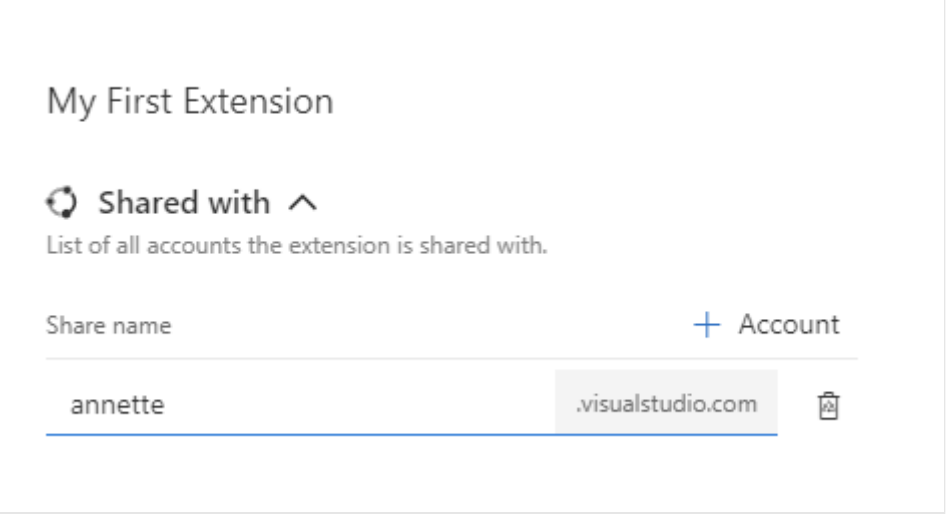
Share your extension

You must share your extension with an organization before you can install it in Azure DevOps. To share an extension, do the following tasks:

1. From the Marketplace management portal, select your extension from the list, right-click, and then choose **Share/Unshare** or **Publish/Unpublish**, depending on the extension.



2. Select **Organization**, and then enter the name of your organization. Select **Enter**.



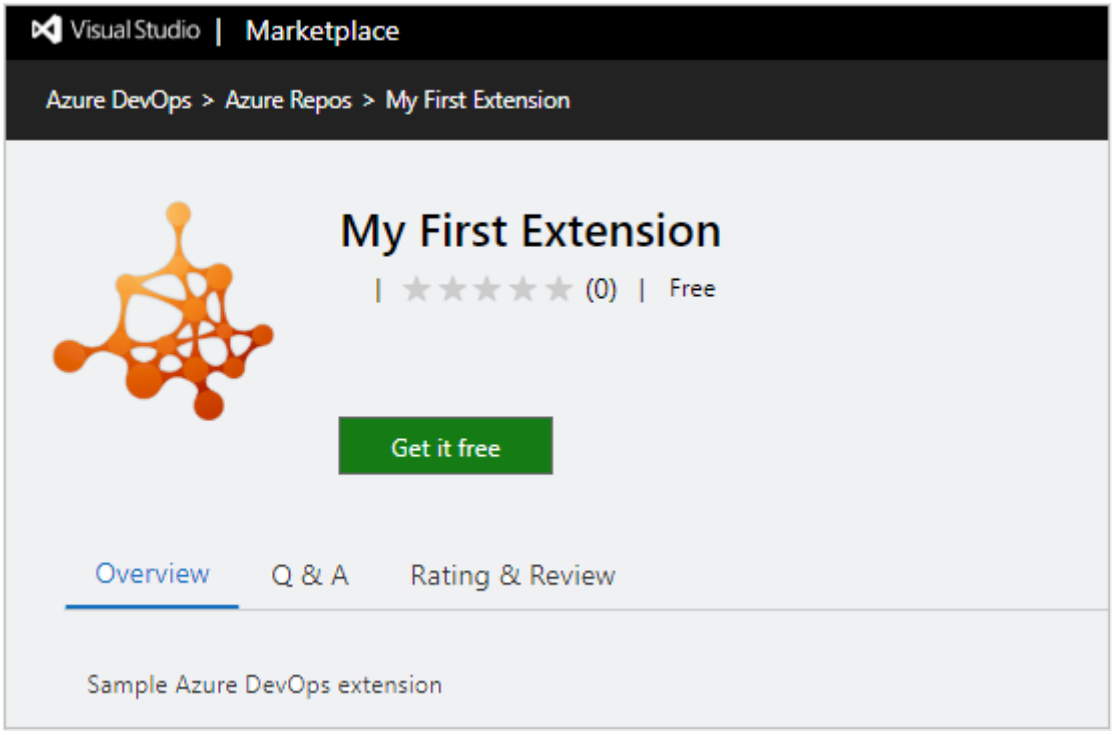
3. Close the panel.

Your extension can now be installed into this organization.

Install your extension

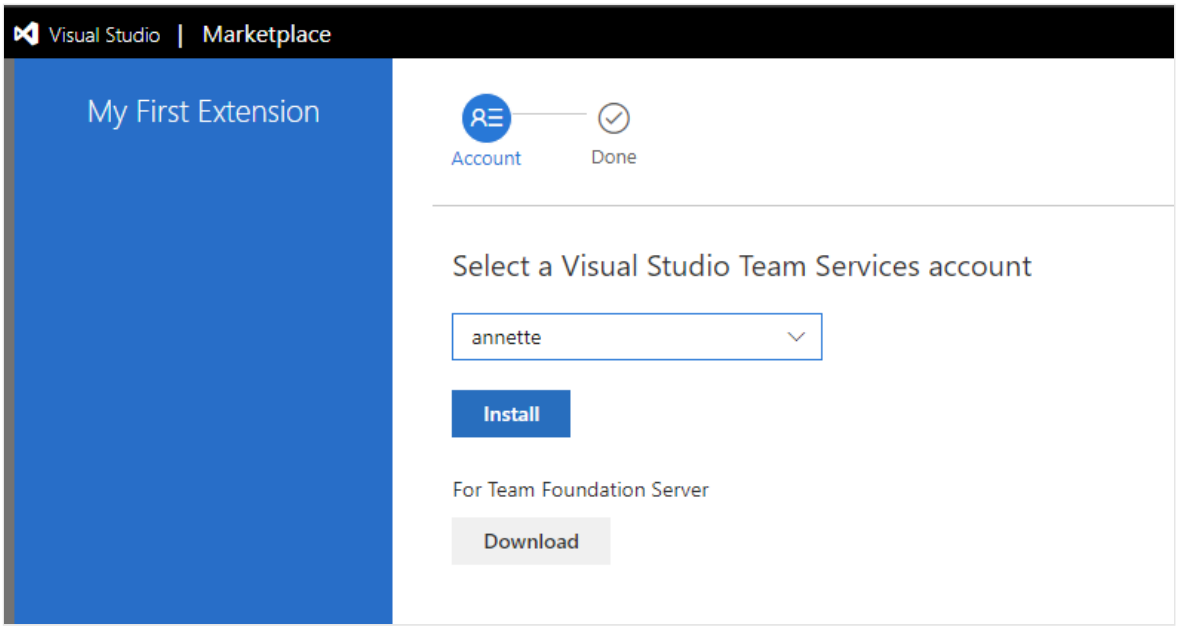
To install your shared extension, do the following steps.

1. In the Marketplace, select your extension to open its overview page.



> ⓘ **Note**
>
> Because your extension is private, only you and any member of the organization it's shared with can see this page.

2. Select **Get it free** to start the installation process. Select the organization you shared the extension with from the dropdown menu.



3. Select **Install**.

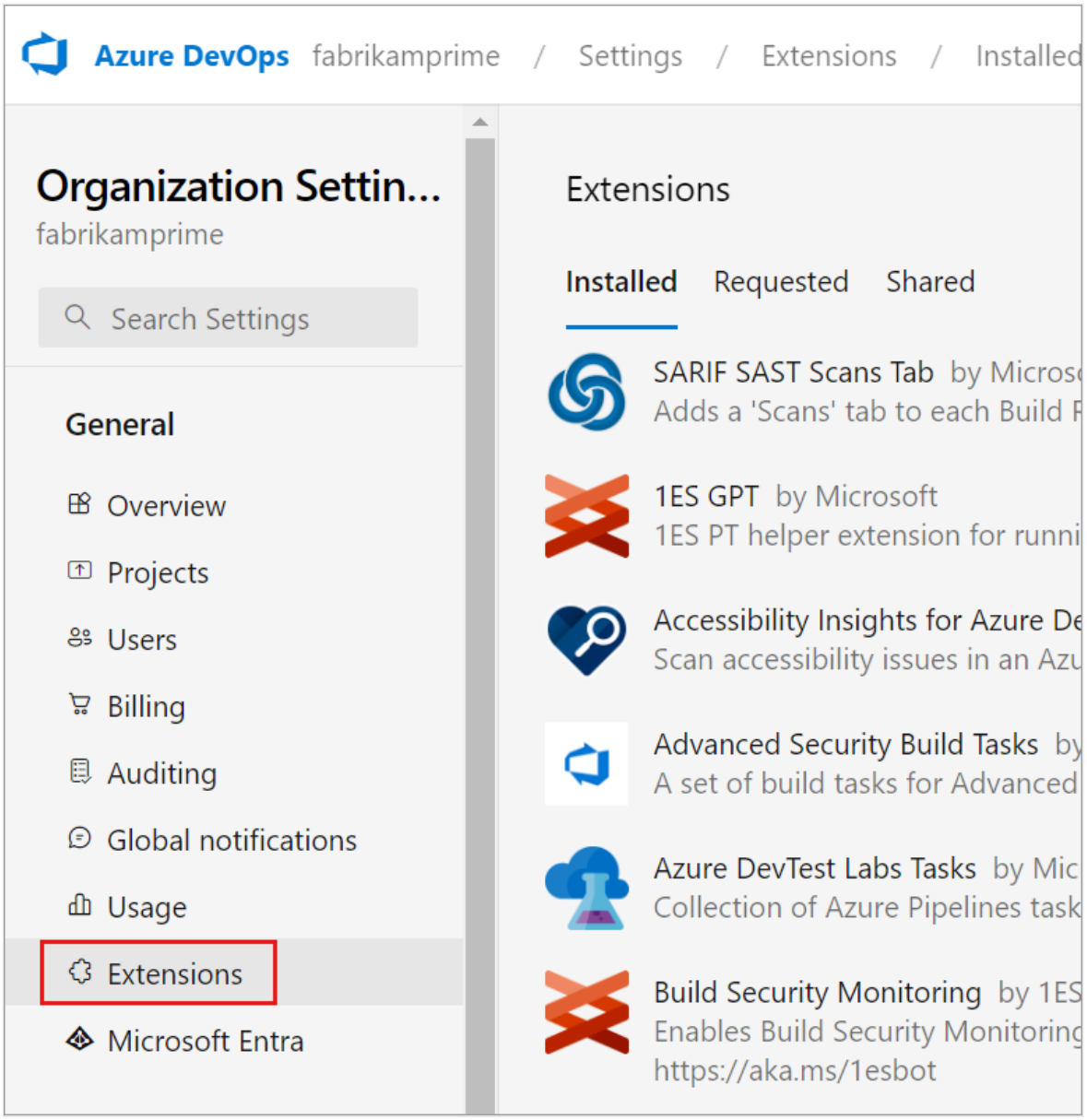
Congratulations! You installed your extension into an organization and you're ready to try it.

Try your extension

1. Select **Proceed to organization** at the end of the installation wizard to go to the home page of the organization the extension was installed to (`https://dev.azure.com/{organization}`).

2. Refresh your browser.

3. Open **Organization settings**, and then select **Extensions**.



You should see the new extension on the **Installed** tab.

Debug your extension

To debug the extension using Visual Studio or Browser Developer Tools, change the manifest by adding the `baseUri` property. This action speeds up the development without the need to redeploy the extension each time you change source code.

JSON

```
{
    ...
    "baseUri": "https://localhost:44300",
    ...
}
```

When you change the manifest, it loads the extension from your local web server instance. For example, IISExpress in Visual Studio. After you change the manifest, deploy and install this debugging extension only once.

> **Note**
>
> Run your local web server in SSL mode because Azure DevOps demands that the web page is served from a secure source. Otherwise, you get an error in the browser console during the extension IFRAME loading.

Update your extension

To change an extension that's already published, update it.

> **Tip**
>
> We recommend updating the extension over removing and re-uploading. We also recommend having two extensions, for example, `publisher.extension` and `publisher.extension-dev`. `Publisher.extension` is public in the Marketplace, where customers can install it in their Azure DevOps organizations. `Publisher.extension-dev` is kept private in the Marketplace and can be shared with an organization that you own and control. You don't need to maintain two copies of source code of the extension. You can maintain two manifest files - one for each extension and during packaging of the extension you can provide the respective manifest file to the tfx-cli tool. For more information on arguments required for the tool, see **TFX extension commands**.

1. Select an extension from the list of displayed items.
2. Right-click and select **Update** for the `publisher.extension-dev`, for example.
3. Validate your extension.
4. Make the same updates to the production version, `publisher.extension`, for example.
5. Browse to the .vsix for your extension and upload it.

The updated version of your extension automatically gets installed to accounts that have it already installed. New accounts where the extension is installed in the future also receive the latest version.

Make your extension public

While you develop your extension or integration for the Marketplace, keep it private. To make your extension available publicly, set the public flag to `true` in your manifest.

Qualifications

To have a public listing on the Marketplace, your integration or extension must meet the following qualifications:

- Works with or extends Azure DevOps.
- You, or your company, own, develop, and are licensed to distribute and advertise the integration or extension.
- The extension or integration is actively maintained.

Microsoft might also request a demo and to review the content planned for your Marketplace entry.

Top Publisher

The Top Publisher program is only available for publishers with Azure DevOps extensions or integrations. It's not applicable for Visual Studio IDE and Visual Studio Code extension publishers.

The Top Publisher program recognizes publishers with commitment to their customers and the Marketplace through exemplary policies, quality, reliability, and support. Once you become a Top Publisher, all of your public offerings display the Top Publisher badge.

Top Publisher requirements

The Top Publisher program in the Marketplace is designed to help you evaluate or acquire Azure DevOps extensions and integrations with confidence. The Top Publisher badge implies that the publisher shows commitment to their customers and the Marketplace through exemplary policies, quality, reliability, and support. It's for

publishers with one or more global Azure DevOps extensions or integrations and isn't applicable for Visual Studio IDE and Visual Studio Code extension publishers.

Marketplace assigns the badge to a publisher after carefully reviewing the publisher across the following parameters:

- Privacy policy
- Licensing policy
- Support policy
- Documentation
- Q & A responsiveness
- Ratings & review for their offerings
- Active uptake and the install count for their offerings
- Manage at least one public Azure DevOps extension from the publisher.
- The public extension should have more than 5000 installs, with an active install count exceeding 1000.

You can expect timely support and a good overall experience when you get an extension from a Top Publisher. Check out the offerings from the Top Publishers.

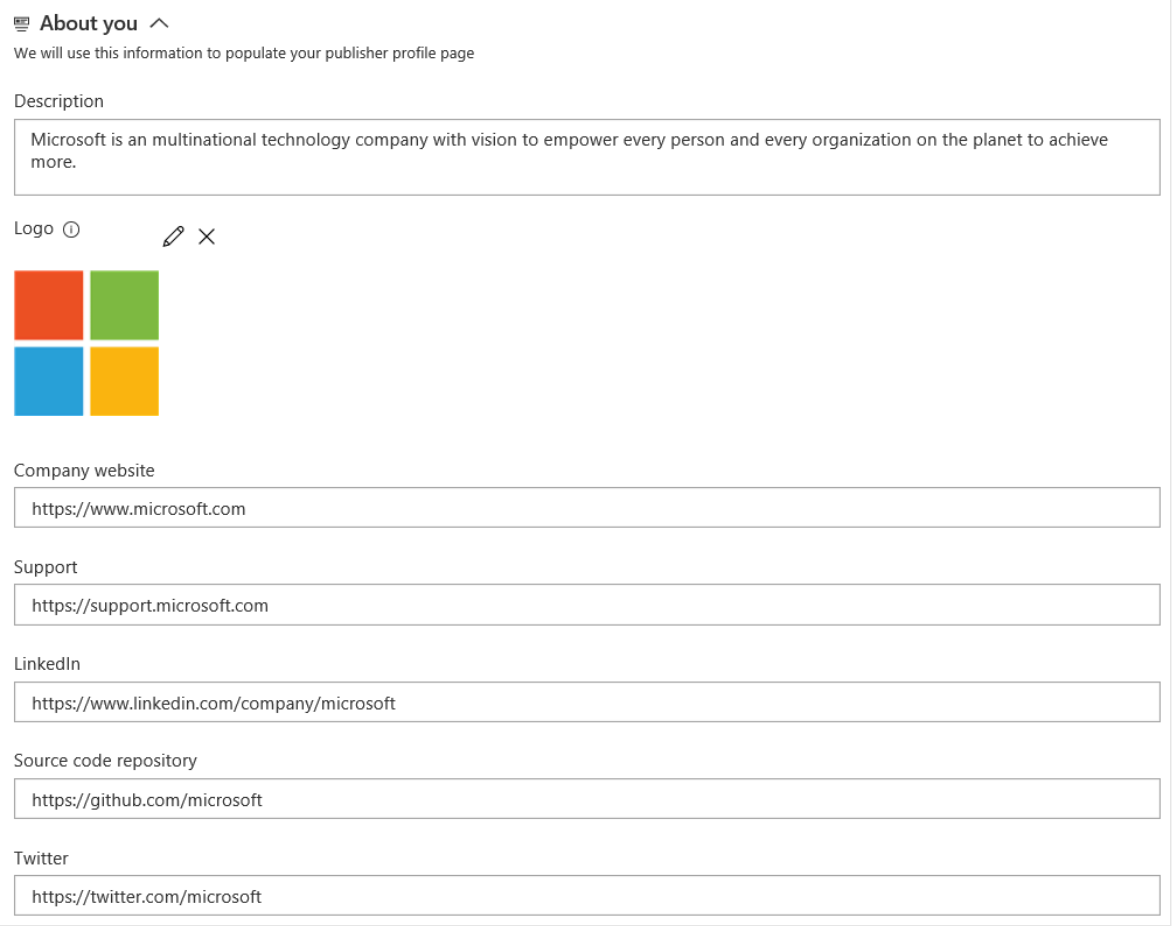
For more information on adding policies to your offering, see the extension manifest.

1. Update your publisher profile.

   Through the publisher profile, you can showcase all of your offerings in one place along with key publisher-related information. To provide the information, which shows up in the profile, do the following steps:

   a. Sign in to https://marketplace.visualstudio.com/manage/publishers using the account with which you publish and manage your offerings in the Visual Studio Marketplace.

b. Select the publisher and complete the **About you** section in the **Details** tab.



c. Save your changes and select **View profile** to see how it appears to consumers. You can use this profile page to evangelize your offering(s).

> **Note**
>
> Through this program, it is the publisher that is being certified. This doesn't cover the software or security of their extensions and integrations. We recommend you be aware of the safety information when you're evaluating the offerings from a publisher.

If you got an extension from a Top Publisher and aren't satisfied with your experience, consider engaging with the publisher first.

Apply to be a Top Publisher

1. Sign in to https://marketplace.visualstudio.com/manage/publishers using the account with which you publish and manage your offerings in Marketplace
2. Select the publisher and navigate to its **Top Publisher** tab. Note: you need to have one or more global Azure DevOps (Server/Service) extension or integration for the tab to appear.

3. If you meet part of the previously listed requirements and are the publisher's owner, you see an option to apply for the program. On application, an email is sent to the Marketplace team to review your case. They respond in under 10 business days with next steps, clarifying questions or with the grant of the badge.

The team likely looks at other parameters, such as active uptake of your offerings, install/get started counts and ratings & reviews across your offerings before granting the badge. Microsoft reserves the right to grant, reject or revoke the Top Publisher badge at any time.

Once a publisher is a Top Publisher, then all its future updates and offerings must meet the previously listed requirements.

Respond to Marketplace extension reviews

You can respond to reviews that customers leave for your extensions in the Visual Studio Marketplace. Find and select **Reply** next to a review if you have one of the following permissions: owner, creator, or contributor.

You can leave only one response. Avoid using reviews as a support forum. If you need more details, please provide a support alias for the reviewer to contact. You can then resolve their problems externally and update your reply with a resolution.

Guidelines for publisher responses

Keep the Visual Studio Marketplace an open, inviting, respectful, and helpful place for customers to find, try, install, and review extensions. Communication plays an important role in keeping a healthy community. To help create this environment, here are guidelines for publishers responding to customer reviews. Think deeply about your customer interactions and reflect on the spirit of the customer experience that the Marketplace is trying to create.

- Reviews are reserved for customer comments. Use *Reply* only to respond to a review.
- Reviews are for sharing customer opinions, so *all opinions are valid*. Customers are entitled to their opinions, so treat comments respectfully as feedback without debate, criticism, or argument.
- Make sure that your responses add value and are relevant to your customers' comments.
- Focus on precisely addressing questions or problems. If you need more details, ask the customer to contact you over email, rather discuss in reviews. When you

resolve the problem, update your reply with the resolution. You can edit your reply, just like customers can edit their reviews.
- If you come across any inappropriate reviews, like spam, abusive, or offensive content, for any extension, flag it for our review.

Request to void a review

As a publisher, you can appeal to void a review if the issue reported is because of the Marketplace or underlying platform. If the issue is valid, Marketplace admins void the rating. You can **Appeal** from ratings and review section on your extension hub page.

Unpublish an extension

You can unpublish free extensions, if you no longer want to offer them in the Marketplace.

The following scenarios cover when you might want to remove your extension from the Marketplace:

- You developed another extension and no longer want to offer the current one.
- Your extension has a problem, so you want to remove your extension from the Marketplace until you resolve the problem.
- You published your extension as public by mistake.

Certain criteria must be met for an extension to be unpublished or removed:

Expand table

Action	Requirements
Unpublish	Only **free extensions** might be unpublished.
Remove	Your extension must have **zero (0)** installs to be removed.

> **Important**
>
> If you must remove your extension because of legal or security problems, contact Customer Support at the Developer Community. We review the request and manually delete the extension.

1. Select the extension on your publisher page and choose **Unpublish** on the menu.

Your extension is unpublished immediately from the Marketplace, and new users can't install it. Ratings and reviews for your extension stay intact.

To offer your extension again in the Marketplace, choose **Publish** on the menu.

You can also choose to remove your extension completely from the Marketplace if your extension has zero (0) installs. To do so, choose **Remove** on the menu. This action can't be undone.

Related articles

- Develop a web extension
- Extensibility points

Feedback

Was this page helpful? Yes No

Provide product feedback

Package and publish an integration to the Marketplace

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Do you have a tool, service, or product that integrates with Azure DevOps or Team Foundation Server (TFS)? If so, help users find it by publishing it on the Visual Studio Marketplace. The Marketplace is a one-stop-shop for individuals and teams to find tools that extend and enhance the experience.

Browse the Marketplace to see examples of other integrations and extensions.

> **Note**
>
> If you're looking for packaging and publishing information for extensions, check out **Package & Publish Extensions**.

Publishing requirements

The following list of **requirements** must be met before you publish to the Marketplace.

- Install the extension packaging tool (TFX). Run `npm install -g tfx-cli` from a command prompt.
- Ensure the proper permissions are granted to use any images, for example, icons, logos, screenshots, and so on.
- Include a thorough `overview.md` file to describe your listing in the Marketplace.
- Include an icon for your extension, which is at least 128x128 pixels in size.
- When you're referring to Microsoft products, use full names in place of abbreviations, for example, Azure DevOps vs. AzDO or - any other abbreviation.
- Refrain from using brand names in the name of your extension.

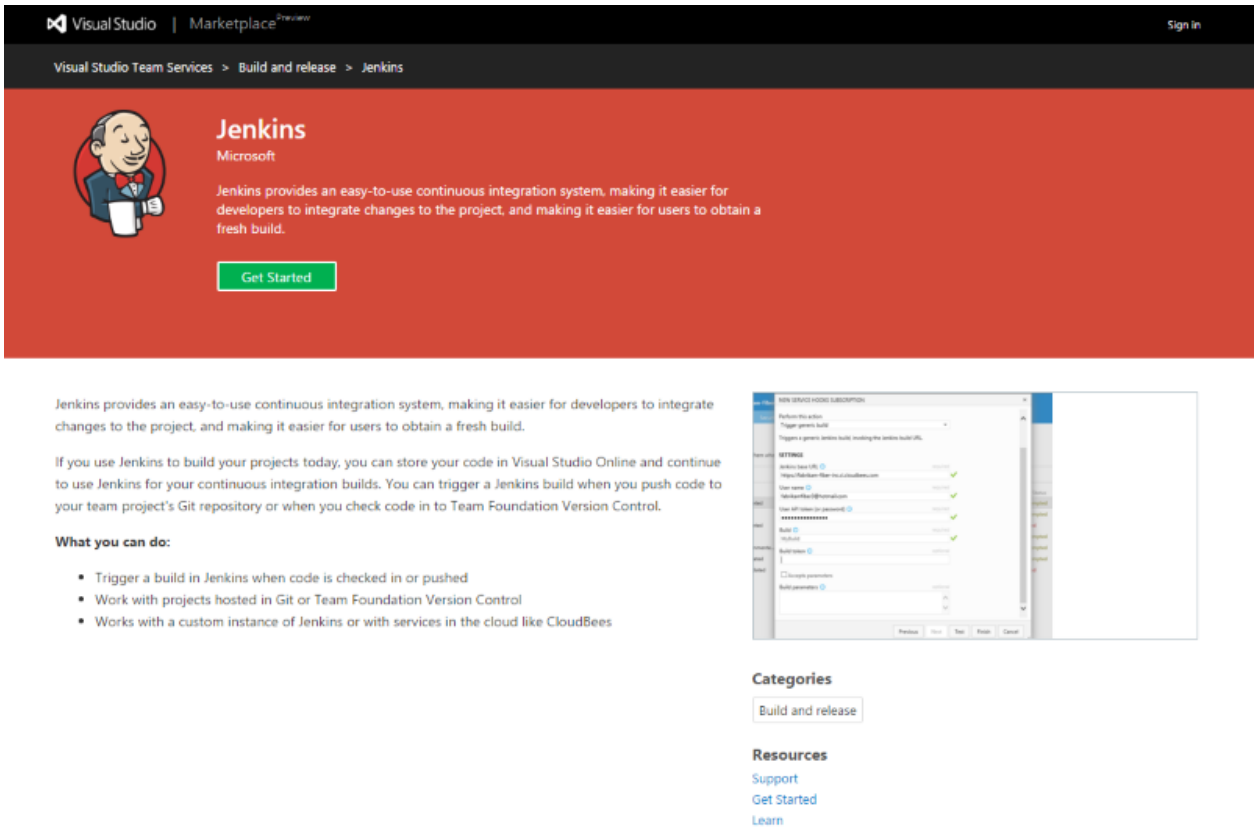
What you need

1. 128x128 pixel logo (PNG or JPEG format) representing your integration, yourself, or your company/organization
2. Minimum of one screenshot showing your integration

3. Call to action / get started URL (where users should go to get started with your integration)

Steps

Publishing to the Marketplace is an iterative process that starts with creating a manifest file that defines your integration and key discovery characteristics (like screenshots, logos, and overview content). This information is used to present your integration to users on the Marketplace, for example:



[Jenkins for Azure DevOps](#)

Note: The term, `extension`, is used in documentations referenced below. Extensions are another type of Marketplace item and share many similarities from a discovery standpoint as integrations.

> Need help with getting your integration on the Marketplace? **Contact us**. And, yes, this e-mail address is monitored by real people.

Create a publisher

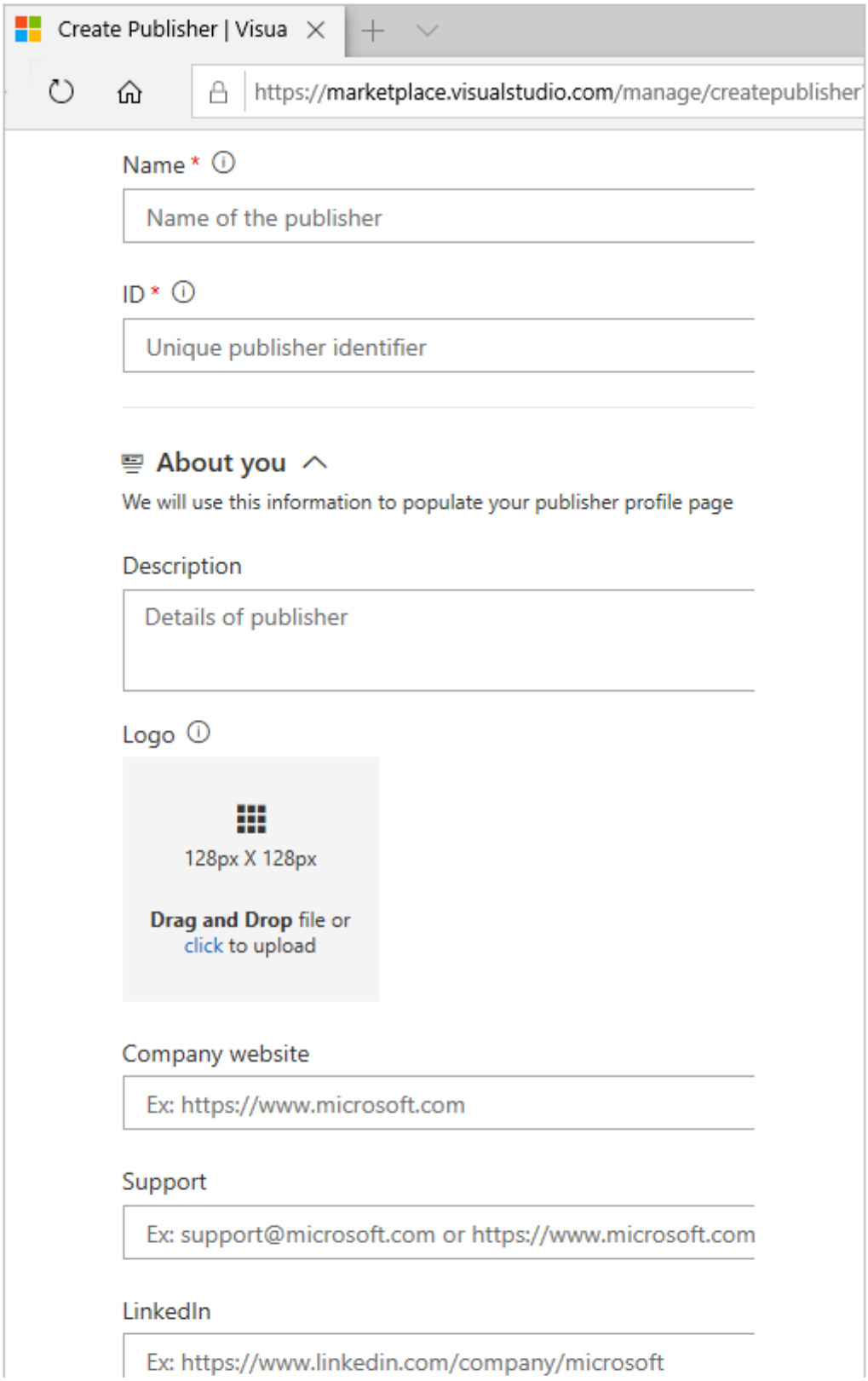
All extensions and integrations, including extensions from Microsoft, have a publisher. Anyone can create a publisher and publish extensions under it. You can also give other

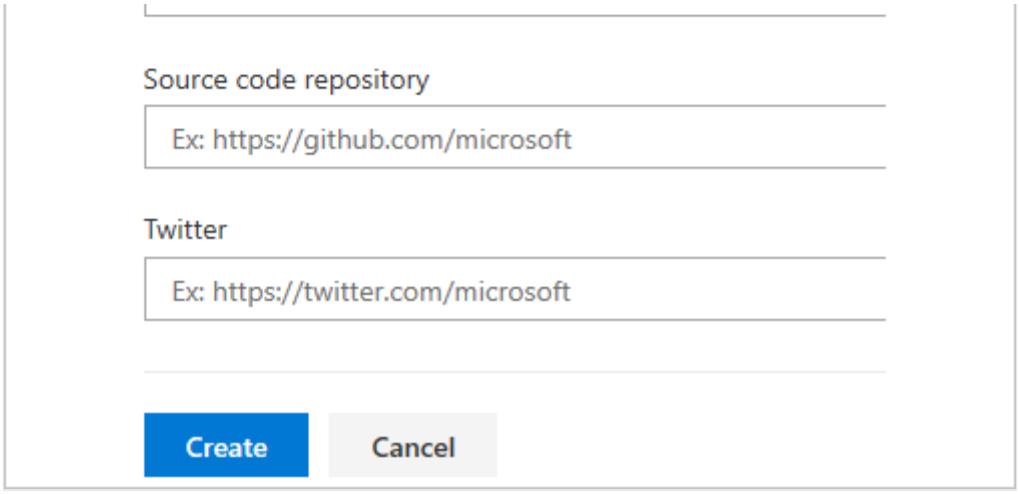
people access to your publisher if a team is developing the extension.

A user owns the publisher, typically the user who created it. The publisher can also be shared with other users.

1. Sign in to the Visual Studio Marketplace Publishing Portal.

2. If you're not already a member of an existing publisher, + **Create a publisher**.
   a. Enter a name in the publisher name field. The ID field should automatically get set based on the name you entered.



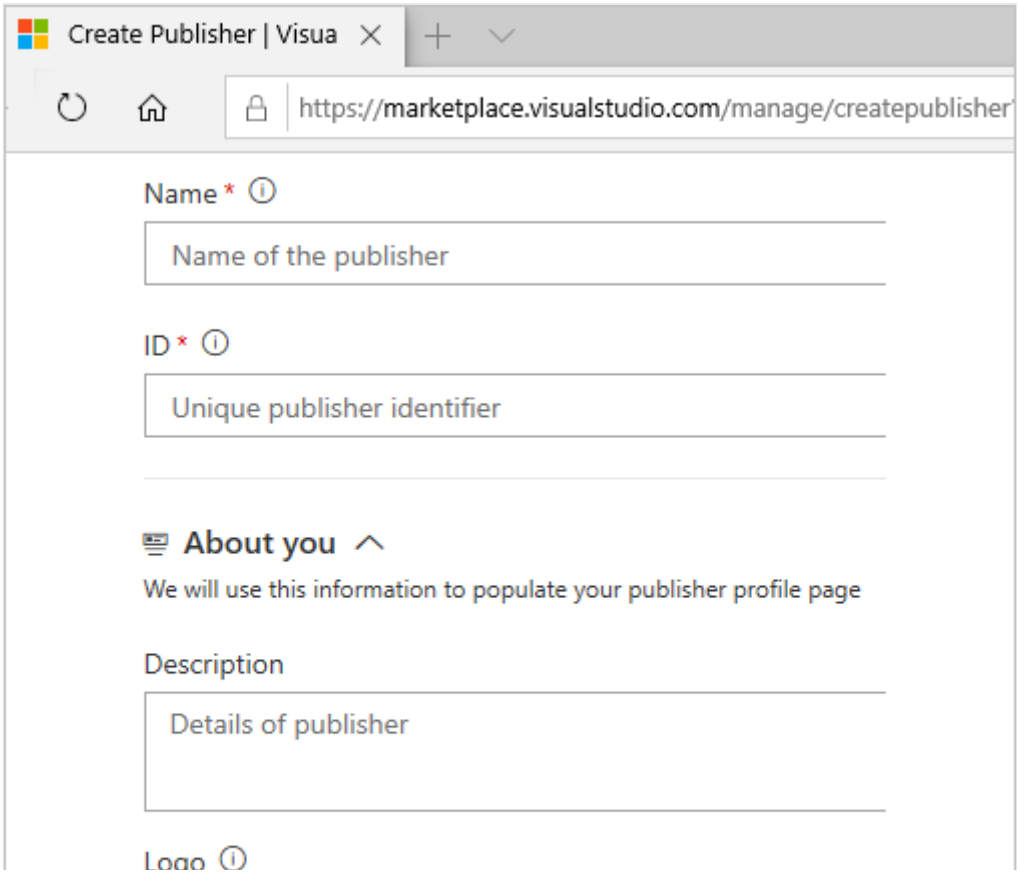


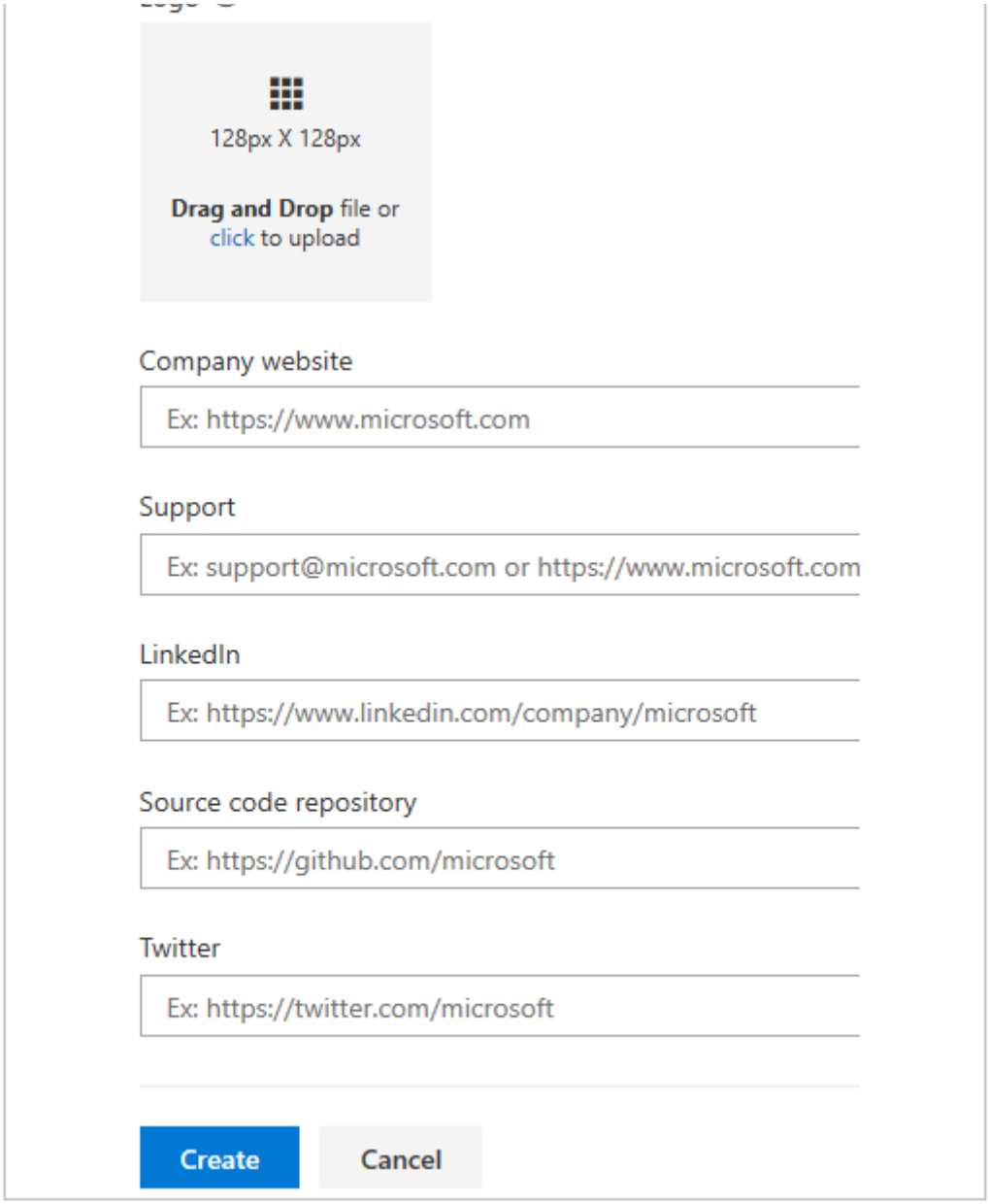
> **Note**
>
> Make note of the ID. You need to set it in the manifest file of your extension.

If you're not prompted to create a publisher, scroll down to the bottom of the page and select **Publish extensions** below *Related sites*.

- Specify an identifier for your publisher, for example: `mycompany-myteam`
  - This identifier is used as the value for the `publisher` attribute in your extension manifest file.
- Specify a display name for your publisher, for example: `My Team`

3. Review the Marketplace Publisher Agreement, and then select **Create**.





Once the publisher's create, you're directed to manage items, but there aren't any items.

Create a folder to contain your item manifest and other assets

Before you package your integration as an extension, you'll need to create a `home` folder to contain some required assets, within this folder:

1. Create a folder called `images` to contain:

    - Logo for your integration (128x128 pixels)
    - Screenshots (1366x768 pixels)

2. Create a file called `overview.md`

    - Describe your integration here
    - To learn more about Markdown, see GitHub Flavored Markdown

3. Create a file called `vss-integration.json`

    - This file is your Marketplace listing's manifest file, it contains many properties to describe your extension in your Marketplace listing. You can browse the extension manifest reference here

Extension Manifest

1. Fill your `vss-integration.json` file with the following JSON:

JavaScript

```
{
    "manifestVersion": 1,
    "id": "myservice",
    "version": "1.0.0",
    "name": "My Service",
    "publisher": "mycompany",
    "description": "Awesome tools to help you and your team do great
things everyday.",
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services.Integration"
        }
    ],
    "icons": {
        "default": "images/service-logo.png"
    },
    "categories": [
        "Plan and track"
    ],
    "tags": [
        "working",
        "people person",
        "search"
    ],
    "screenshots": [
        {
            "path": "images/screen1.png"
        },
        {
            "path": "images/screen2.png"
        }
    ],
    "content": {
        "details": {
            "path": "overview.md"
        },
        "license": {
            "path": "fabrikam-license-terms.md"
        }
```

```
    },
    "links": {
        "getstarted": {
            "uri": "https://www.mycompany.com/help/getstarted"
        },
        "learn": {
            "uri": "https://www.mycompany.com/features"
        },
        "support": {
            "uri": "https://www.mycompany.com/support"
        }
    },
    "branding": {
        "color": "rgb(34, 34, 34)",
        "theme": "dark"
    }
}
```

2. Update the JSON using the following reference:

These properties are required:

Property	Description	Notes
manifestVersion	*A number corresponding to the version of the manifest format.*	Should be `1`.
ID	*The extension's identifier.*	Th ID is a string that must be unique among extensions from the same publisher. It must start with an alphabetic or numeric character and contain 'A' through 'Z', 'a' through 'z', '0' through '9', and '-' (hyphen). Example: `sample-extension`.
version	*A string specifying the version of an extension.*	Should be in the format `major.minor.patch`, for example `0.1.2` or `1.0.0`. You can also add a fourth number for the following format: `0.1.2.3`
name	*A short, human-readable name of the extension. Limited to 200 characters.*	Example: `"Fabrikam Agile Board Extension"`.
publisher	*The identifier of the publisher.*	This identifier must match the identifier the extension is published under. See Create and manage a publisher.

Property	Description	Notes

categories	*Array of strings representing the categories your extension belongs to. At least one category must be provided and there's no limit to how many categories you may include.*	Valid values: `Azure Repos`, `Azure Boards`, `Azure Pipelines`, `Azure Test Plans`, and `Azure Artifacts`.
		
		Notes:
		• Use version >=0.6.3 of the tfx-cli if you're publishing the extension programmatically.
		• If you're using Azure DevOps Extension Tasks extension to publish, ensure that its version is >= 1.2.8. You might have to approve the extension update because of recent scope changes.
		• The categories previously mentioned are natively present in Visual Studio Marketplace and Azure DevOps Server 2019 & above. For extensions targeting earlier versions of TFS:
		  ◦ If TFS customers acquire your extension via Visual Studio Marketplace (not local gallery) in connected context, then use the categories that are previously stated.
		  ◦ If you are going to share the extension directly (that is, not via the Visual Studio Marketplace) with a customer using TFS <=2018 then use the following categories instead: Code, Plan and track, Build and release, Test, Collaborate, and Integrate. If you need to share both via Visual Studio Marketplace and directly with a TFS <= 2018 customer then you would need to have 2 extension packages.

targets	*The products and services supported by your integration or extension.* For more information, see installation targets.	An array of objects, where each object has an `id` field indicating one of the following: • `Microsoft.VisualStudio.Services` (extensions that works with Azure DevOps or TFS), • `Microsoft.TeamFoundation.Server` (extension that works with TFS), • `Microsoft.VisualStudio.Services.Integration` (integrations that works with Azure DevOps or TFS), • `Microsoft.TeamFoundation.Server.Integration` (integrations that work with TFS)

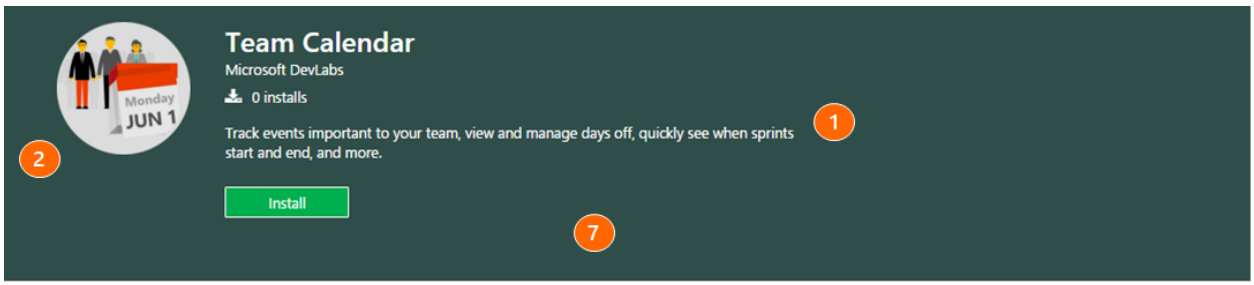
These optional properties help users discover and learn about your extension:

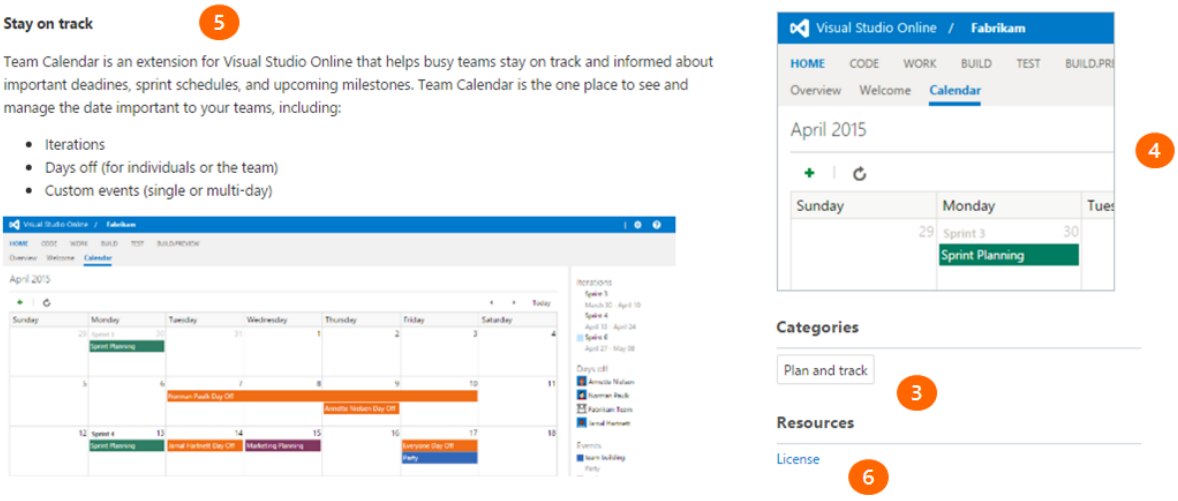
Property	Description	Notes
description	*A few sentences describing the extensions. Limited to 200 characters.*	The description should be your extension's "elevator pitch" - a couple of lines to describe your extension in the Marketplace and make people want to install it. See the example below
icons	*Dictionary of icons representing the extension.*	Valid keys: `default` (128x128 pixels) of type BMP, GIF, EXIF, JPG, PNG and TIFF). Other keys such as `large` (512x512 pixels) may be supported in the future. The value of each key is the path to the icon file in the extension
tags	*Array of string tags to help users find your extension.*	Examples: `agile`, `project management`, `task timer`, and so on.
screenshots	*Array of images that couldn't be included in your **content**.*	Screenshots are more valuable when featured in your **content**, and should be used there to help make a quality market details page for your extension. Use **screenshots** for less important images not featured in your **content**. Each image should be 1366x768 pixels. The `path` of each item is the path to the file in the extension.
content	*Dictionary of content files that describe your extension to users.*	*Every* extension should include solid content. This is how you'll show users what your extension can do. Make it rich, consumable, and include screenshots where necessary. Include an `overview.md` file as your base content piece. Each file is assumed to be in GitHub Flavored Markdown format. The `path` of each item is the path to the Markdown file in the extension. Valid keys: `details`. Other keys may be supported in the future.
links	*Dictionary of links that help users learn more about your extension, get support, and move.*	Valid keys: `getstarted` - first steps, how to setup or use. `learn` - deeper content to help users better understand your extension or service. `license` - end-user license agreement. `privacypolicy` - privacy policy for an extension. `support` - get help and support for an extension. The value of each key is an object with a `uri` field, which is the absolute URL of the link
repository	*Dictionary of properties describing the source code repository for the extension*	Valid Keys: `type` - Type of repository. Example: *git*. `uri` - Absolute URL of the repository.

Property	Description	Notes
badges	*Array of links to external metadata badges like TravisCI, Appveyor, and so on, from the approved badges sites*	Valid keys: `href` - Link the user navigates to when selecting the badge. `uri` - The absolute URL of the badge image to be displayed. `description` - Description of the badge, to be displayed on hover.
branding	*Dictionary of brand-related properties.*	Valid keys: `color` - primary color of the extension or publisher; can be a hex (#ff00ff), RGB (rgb(100,200,50)), or supported HTML color names (blue). `theme` - complements the color; use *dark* for dark branding colors, or *light* for lighter branding colors.

Details page

- 1 - description
- 2 - icon
- 3 - categories
- 4 - screenshots
- 5 - content (details)
- 6 - links
- 7 - branding





Make sure the "public" attribute gets set to "false" (or not set at all) to avoid your extension or integration from becoming prematurely visible to all users on the Marketplace.

Package your manifest and assets

Get the package tool (tfx-cli)

You can install or update the Cross-platform CLI for Azure DevOps (tfx-cli) using `npm`, a component of Node.js, from your command line.

```
no-highlight
npm i -g tfx-cli
```

Package your integration in a .vsix file

```
no-highlight
tfx extension create --manifest-globs vss-extension.json
```

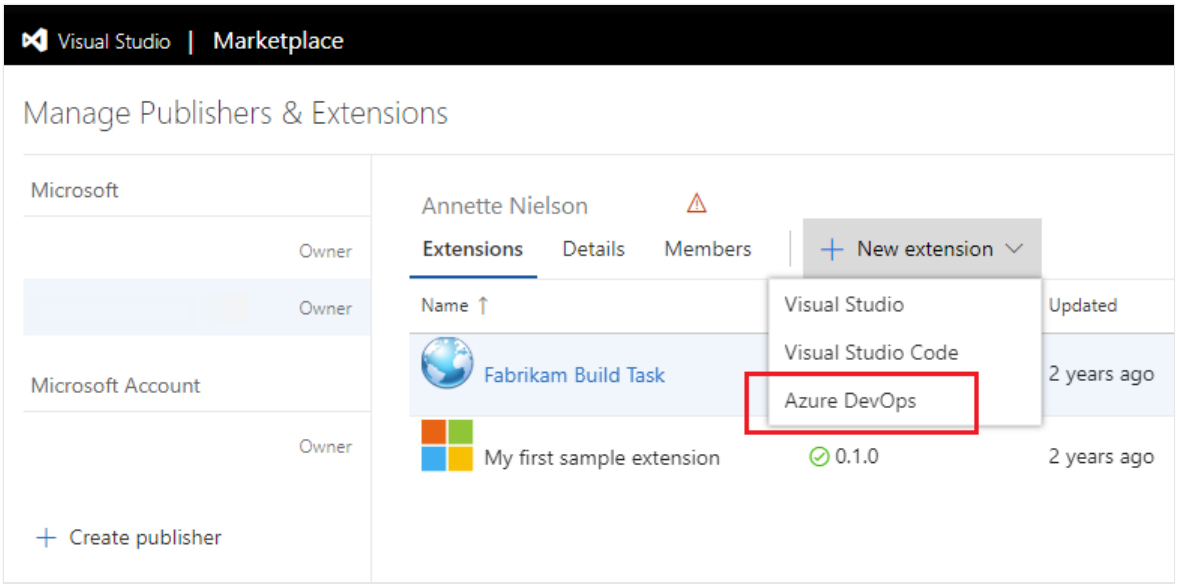
> **Note**
>
> An extension/integration's version must be incremented on every update.
> If you haven't incremented your extension/integration in the manifest, you should pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and saves the new version to your manifest.

Publish your integration to the Marketplace

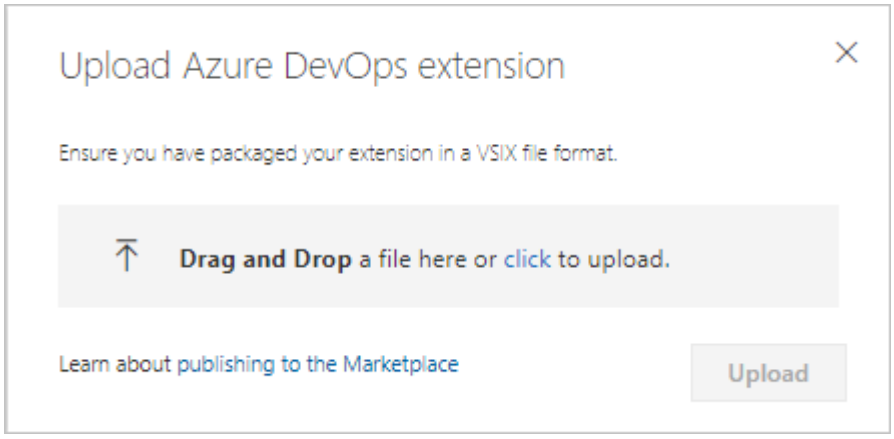
Once your extension is packaged, you can upload it to the Marketplace under a publisher. The `publisher` identifier specified in your extension's manifest file must match the identifier of the publisher the extension is uploaded under.

1. From the management portal, select your publisher from the drop-down menu at the top of the page.

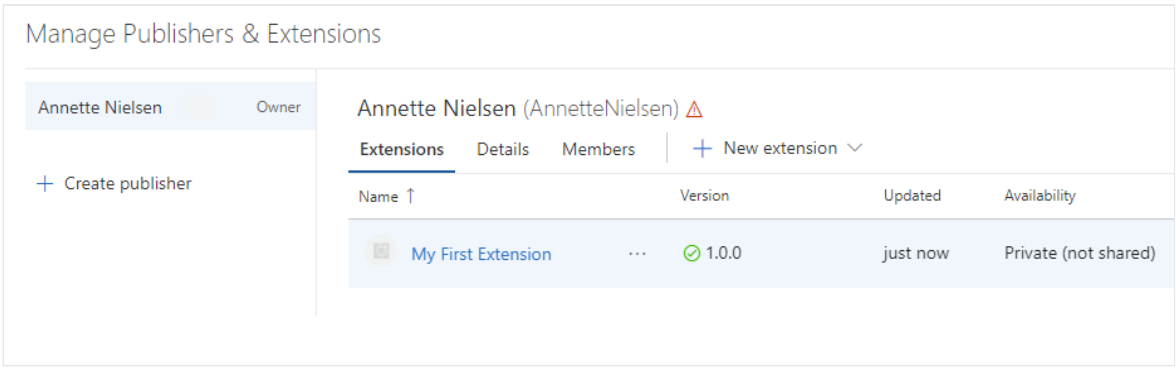
2. Select **New extension** > **Azure DevOps**.



3. Drag and drop your file or select it to find your VSIX file, which you created in the previous packaging step, and then choose **Upload**.



After quick validation, your extension appears in the list of published extensions. Don't worry, the extension is only visible to you.



At this point, your extension isn't visible to any accounts and can't be installed until you share it.

> **Note**

Microsoft runs a virus scan on each new and updated extension package published. Until the scan is all clear, we don't publish the extension in the Marketplace for public usage. This way we also avoid surfacing inappropriate or offensive content on the Marketplace pages.

Share your integration

Before you can install an integration into an organization in Azure DevOps or TFS, you must share it with that organization. Sharing is a requirement during development and testing of an integration, as it's the only way to run an integration.

To share an integration do the following tasks:

1. Select an integration from the list of displayed items
2. Select the **Share** button
3. Specify the name of the organization to make this integration visible to.

    - For example, to make an integration visible to the **dev.azure.com/fabrikam-fiber-inc** organization, specify `fabrikam-fiber-inc`.

Update an item

To change an extension that's already published, update it.

> **Tip**
>
> We recommend updating the extension over removing and re-uploading. We also recommend having two extensions, for example, `publisher.extension` and `publisher.extension-dev`. `Publisher.extension` is public in the Marketplace, where customers can install it in their Azure DevOps organizations. `Publisher.extension-dev` is kept private in the Marketplace and can be shared with an organization that you own and control. You don't need to maintain two copies of source code of the extension. You can maintain two manifest files - one for each extension and during packaging of the extension you can provide the respective manifest file to the tfx-cli tool. For more information on arguments required for the tool, see **TFX extension commands**.

1. Select an extension from the list of displayed items.
2. Right-click and select **Update** for the `publisher.extension-dev`, for example.
3. Validate your extension.

4. Make the same updates to the production version, `publisher.extension`, for example.
5. Browse to the .vsix for your extension and upload it.

The updated version of your extension automatically gets installed to accounts that have it already installed. New accounts where the extension is installed in the future also receive the latest version.

Make your integration public (visible to everyone)

For information on making your integration public, visit Make your listing public.

Package and publish extensions

Article • 01/09/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Once you develop your extension, you can package and publish it to the Visual Studio Marketplace. The Marketplace is a global repository for private and public extensions, integrations, and other offers from Microsoft.

> **Note**
>
> For information on the discovery properties available in your extension's manifest file that helps users discover and learn about your extension, see the Extension Manifest Reference.

Prerequisites

The following list of requirements must be met before you publish to the Marketplace.

- Install the extension packaging tool (TFX). Run `npm install -g tfx-cli` from a command prompt.
- Ensure the proper permissions are granted to use any images, for example, icons, logos, screenshots, and so on.
- Include a thorough `overview.md` file to describe your listing in the Marketplace.
- Include an icon for your extension, which is at least 128x128 pixels in size.
- When you refer to Microsoft products, use full names in place of abbreviations, for example, Azure DevOps vs. AzDO or - any other abbreviation.
- Refrain from using brand names in the name of your extension.

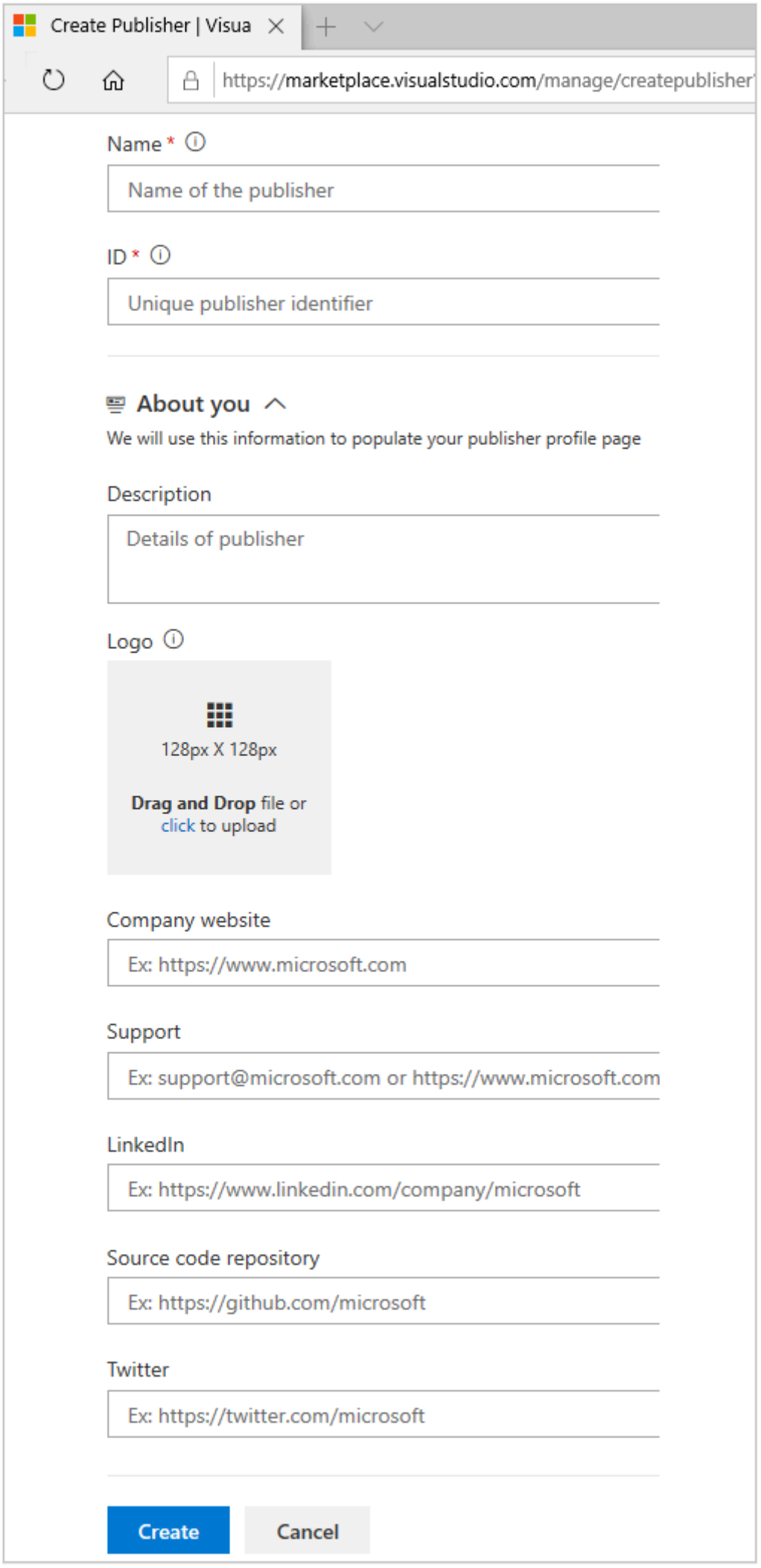
Create a publisher

All extensions and integrations, including extensions from Microsoft, have a publisher. Anyone can create a publisher and publish extensions under it. You can also give other people access to your publisher if a team is developing the extension.

A user owns the publisher, typically the user who created it. You can also share the publisher with other users.

1. Sign in to the Visual Studio Marketplace Publishing Portal.

2. If you're not already a member of an existing publisher, **+ Create a publisher**. Enter a name in the publisher name field. The ID field should automatically get set based on the name you entered.

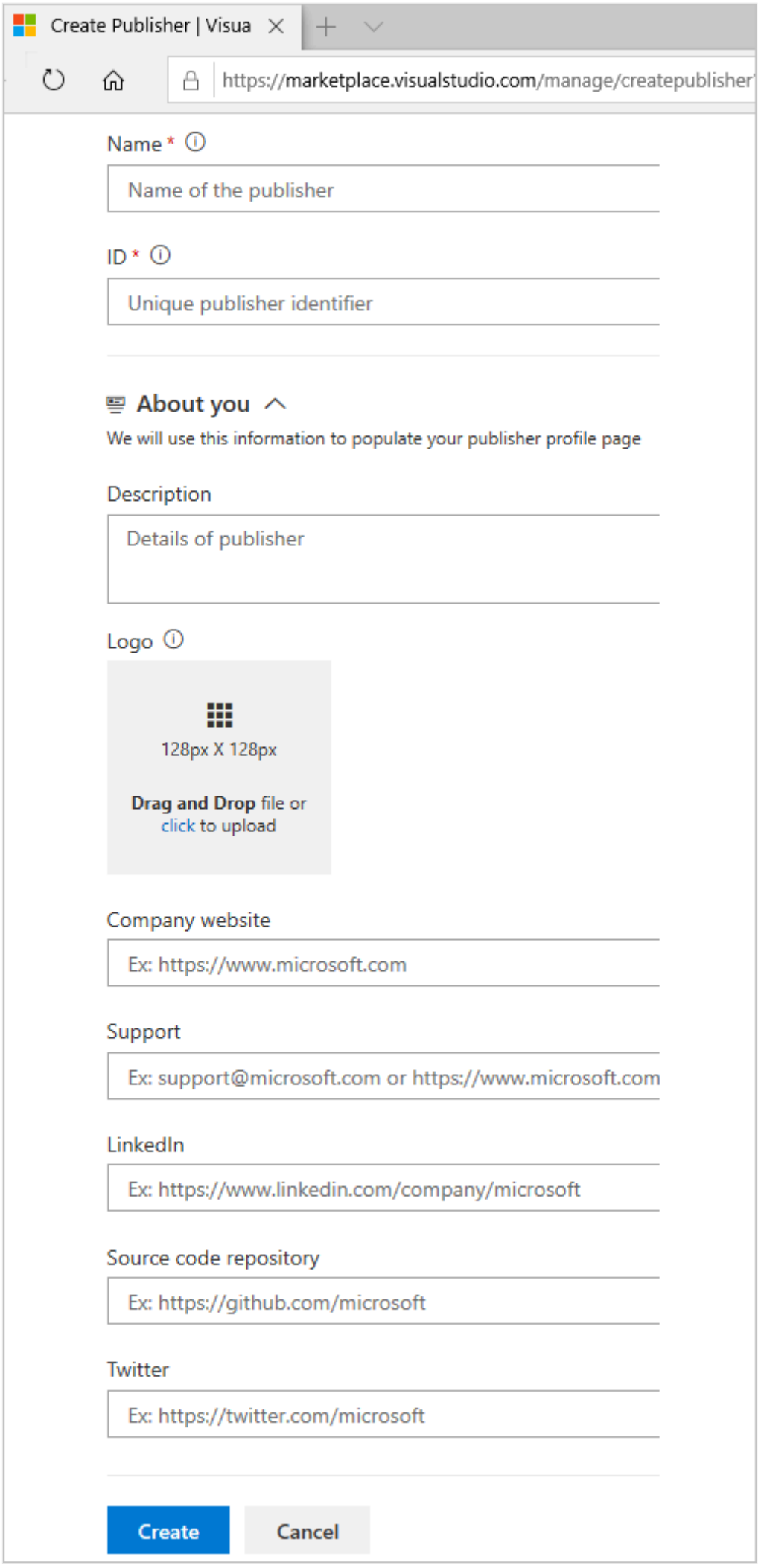

Create Publisher | Visua
https://marketplace.visualstudio.com/manage/createpublisher
Name *
Name of the publisher
ID *
Unique publisher identifier
About you
We will use this information to populate your publisher profile page
Description
Details of publisher
Logo
128px X 128px
Drag and Drop file or
click to upload
Company website
Ex: https://www.microsoft.com
Support
Ex: support@microsoft.com or https://www.microsoft.com
LinkedIn
Ex: https://www.linkedin.com/company/microsoft
Source code repository
Ex: https://github.com/microsoft
Twitter
Ex: https://twitter.com/microsoft
Create
Cancel

> **Note**
>
> Make note of the ID, as you need to set it in the manifest file of your extension.

If you're not prompted to create a publisher, scroll down to the bottom of the page and select **Publish extensions** below *Related sites*.

- Specify an identifier for your publisher, for example: `mycompany-myteam`. This identifier is used as the value for the `publisher` attribute in your extension manifest file.
- Specify a display name for your publisher, for example: `My Team`

3. Review the Marketplace Publisher Agreement, and then select **Create**.


Create Publisher | Visua
https://marketplace.visualstudio.com/manage/createpublisher
Name *
Name of the publisher
ID *
Unique publisher identifier
About you
We will use this information to populate your publisher profile page
Description
Details of publisher
Logo
128px X 128px
Drag and Drop file or
click to upload
Company website
Ex: https://www.microsoft.com
Support
Ex: support@microsoft.com or https://www.microsoft.com
LinkedIn
Ex: https://www.linkedin.com/company/microsoft
Source code repository
Ex: https://github.com/microsoft
Twitter
Ex: https://twitter.com/microsoft
Create
Cancel

Once the publisher's created, you're directed to manage items, but there aren't any items.

Package your extension

To upload your extension, you need to package it as a VSIX 2.0-compatible .vsix file. Microsoft provides a cross-platform command-line interface (CLI) to package and publish your extension.

1. Open your extension manifest file (`vss-extension.json`) and set the value of the `publisher` field to the ID of your publisher. For example:

   JSON

   ```
   {
       ...
       "id": "my-first-extension",
       "publisher": "AnnetteNielsen",
       ...
   }
   ```

2. From a command prompt, run the TFX tool's packaging command from your extension directory.

   ```
   npx tfx-cli extension create
   ```

   A message displays indicating your extension is successfully packaged:

   ```
   === Completed operation: create extension ===
   - VSIX: C:\my-first-extension\AnnetteNielsen.my-first-extension-
   1.0.0.vsix
   - Extension ID: my-first-extension
   - Extension Version: 1.0.0
   - Publisher: AnnetteNielsen
   ```

> **Note**
>
> An extension/integration's version must be incremented on every update.
> If you haven't incremented your extension/integration in the manifest, you should

pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and saves the new version to your manifest.

Check package size

Check the size of the vsix after it gets packaged. If it's greater than 50 MB, you need to optimize it. To do so, see the following considerations:

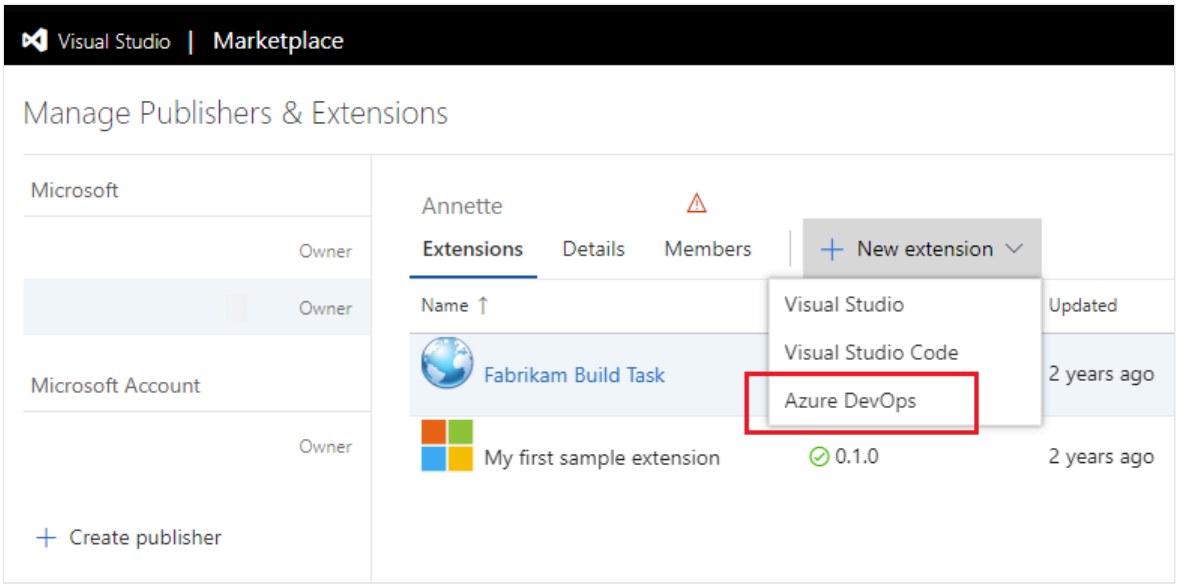
- Deduplicate the common dependencies, if any, by stating them once in the extension package.
- Fetch things at runtime or during install time rather than providing it within the package. Consider using the tool installer lib to pull tool dependencies at runtime. Using the lib offers benefits where the tool gets cached by version so for private agents, it doesn't download every build. We made it a lib so it can be used outside of tool installer tasks. But, the task doesn't work in disconnected scenarios (no internet), which should be in the description / docs for the task.
- Some users have success with WebPack to tree shake their dependencies in their tasks.

Publish your extension

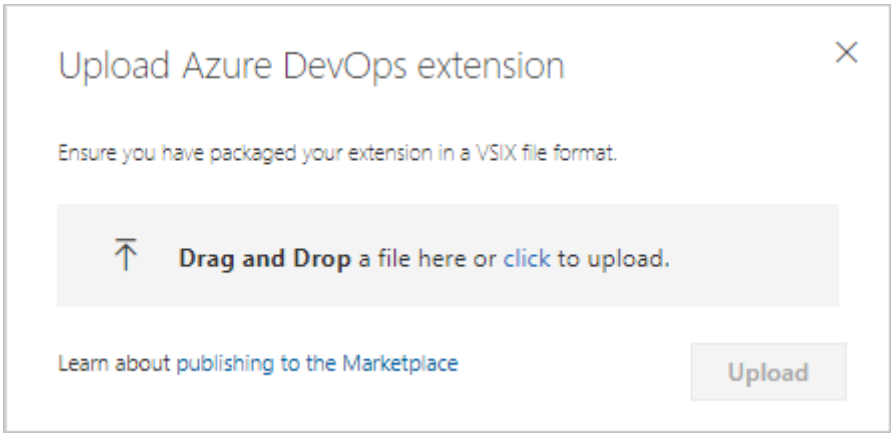
Once your extension is packaged, you can upload it to the Marketplace under a publisher. The `publisher` identifier specified in your extension's manifest file must match the identifier of the publisher the extension is uploaded under.

1. From the management portal, select your publisher from the drop-down menu at the top of the page.

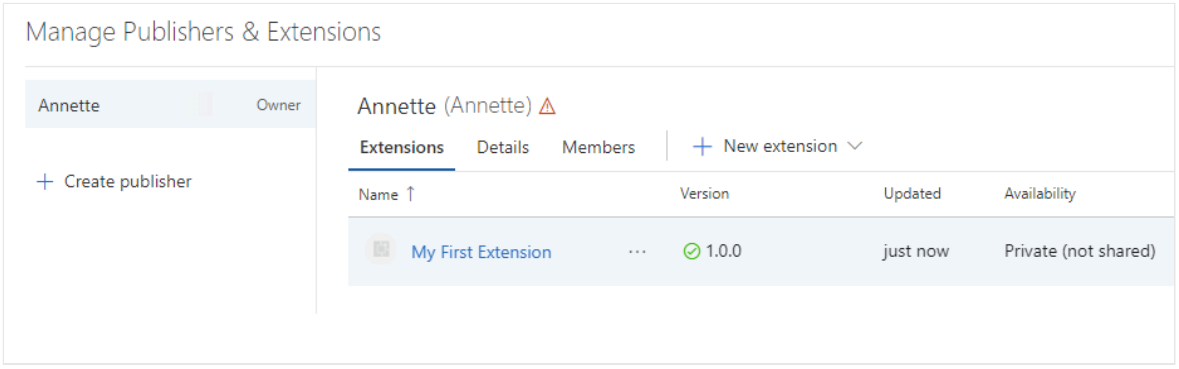
2. Select **New extension** > **Azure DevOps**.



3. Drag and drop your file or select it to find your VSIX file, which you created in the previous packaging step, and then choose **Upload**.



After quick validation, your extension appears in the list of published extensions. Don't worry, the extension is only visible to you.



At this point, your extension isn't visible to any accounts and can't be installed until you share it.

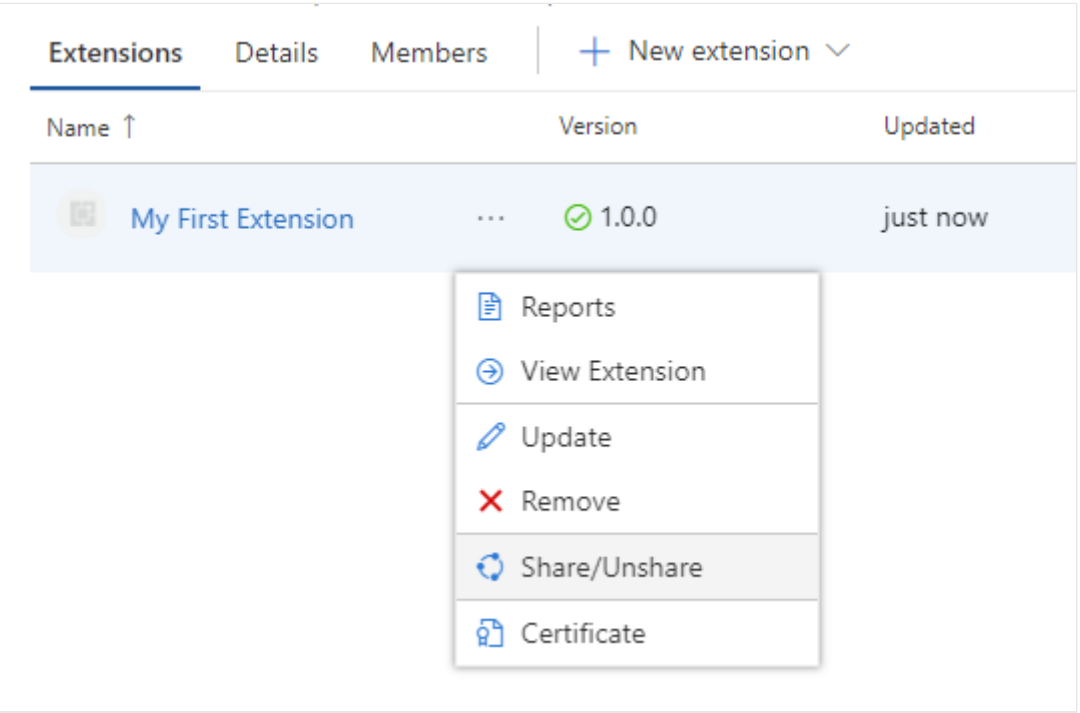
> **Note**

Microsoft runs a virus scan on each new and updated extension package published. Until the scan is all clear, we don't publish the extension in the Marketplace for public usage. This way we also avoid surfacing inappropriate or offensive content on the Marketplace pages.

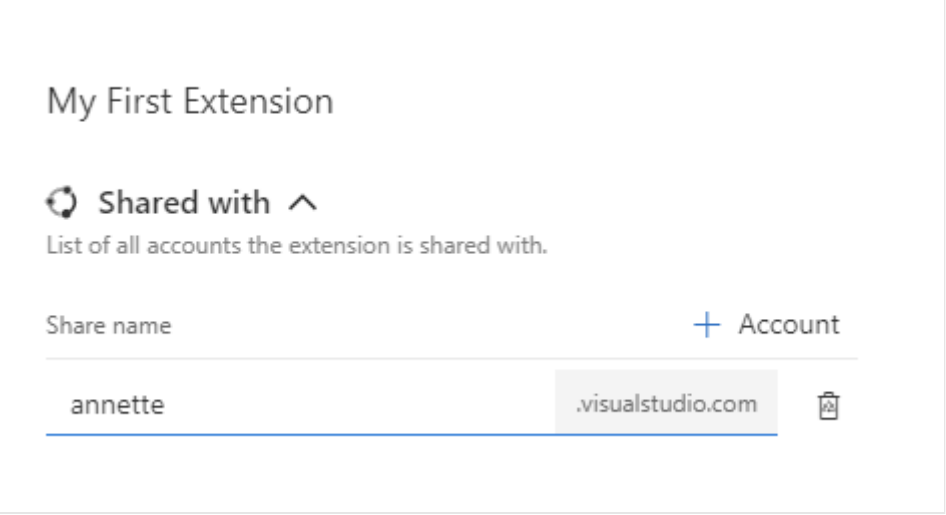
Share your extension

You must share your extension with an organization before you can install it in Azure DevOps. To share an extension, do the following tasks:

1. From the Marketplace management portal, select your extension from the list, right-click, and then choose **Share/Unshare** or **Publish/Unpublish**, depending on the extension.



2. Select **Organization**, and then enter the name of your organization. Select **Enter**.



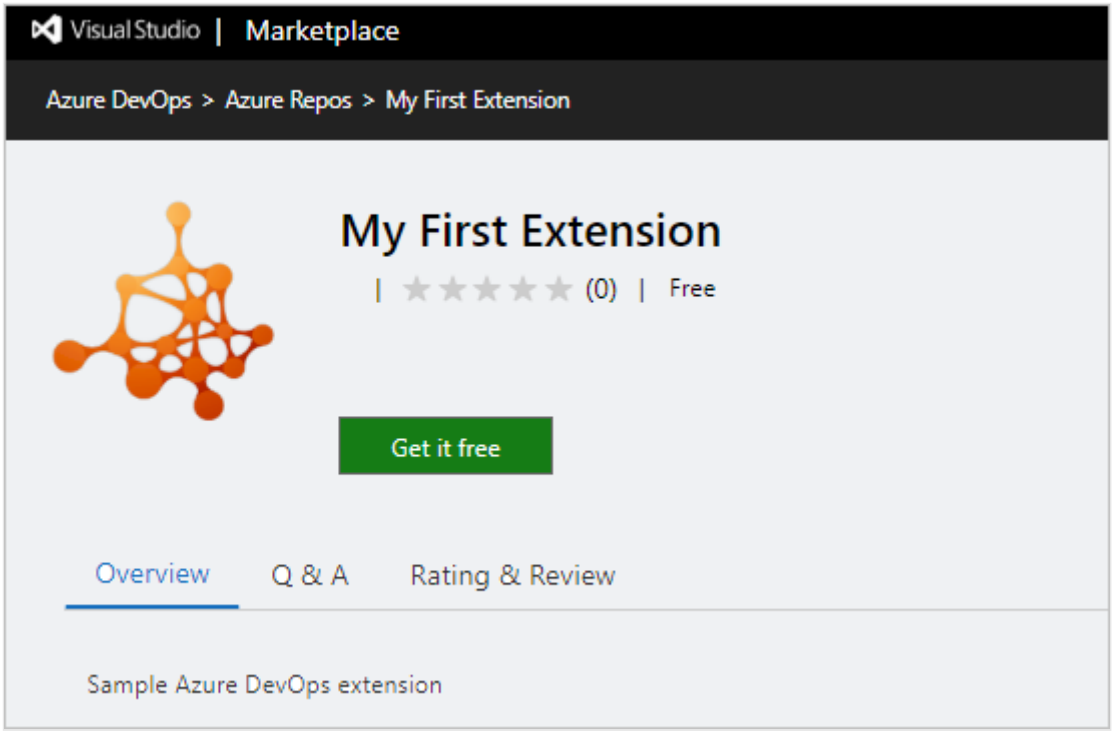
3. Close the panel.

Your extension can now be installed into this organization.

Install your extension

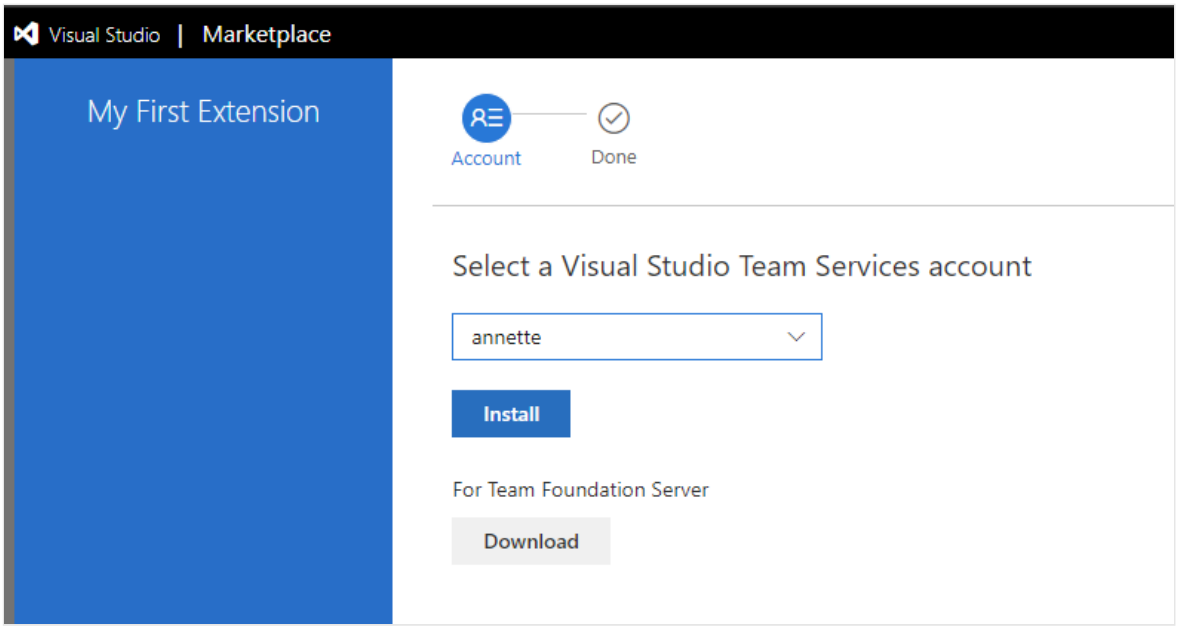
To install your shared extension, do the following steps.

1. In the Marketplace, select your extension to open its overview page.



> ⓘ **Note**
>
> Because your extension is private, only you and any member of the organization it's shared with can see this page.

2. Select **Get it free** to start the installation process. Select the organization you shared the extension with from the dropdown menu.



3. Select **Install**.

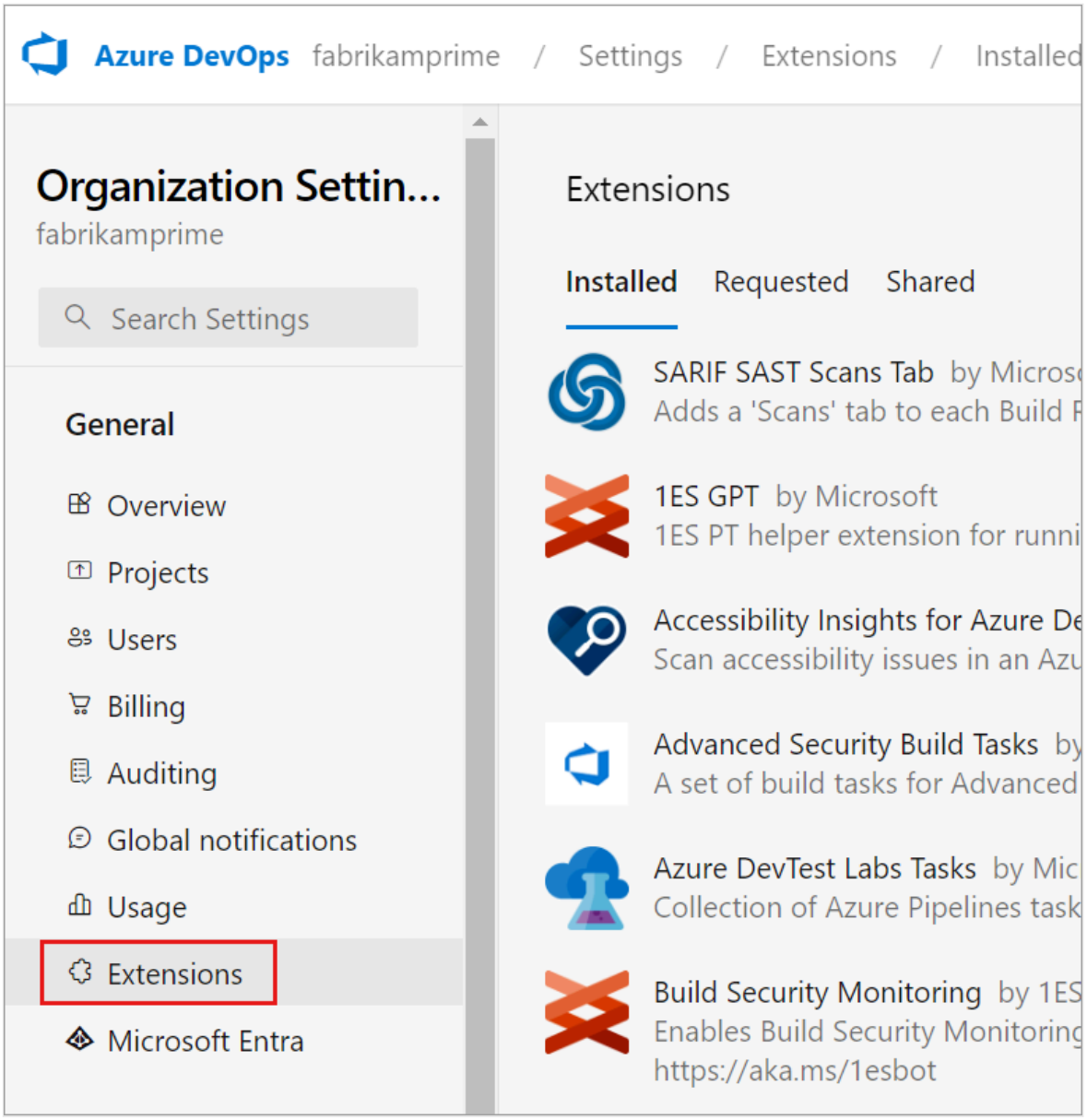
Congratulations! You installed your extension into an organization and you're ready to try it.

Try your extension

1. Select **Proceed to organization** at the end of the installation wizard to go to the home page of the organization the extension was installed to (`https://dev.azure.com/{organization}`).

2. Refresh your browser.

3. Open **Organization settings**, and then select **Extensions**.



You should see the new extension on the **Installed** tab.

Debug your extension

To debug the extension using Visual Studio or Browser Developer Tools, change the manifest by adding the `baseUri` property. This action speeds up the development without the need to redeploy the extension each time you change source code.

JSON

```
{
    ...
    "baseUri": "https://localhost:44300",
    ...
}
```

When you change the manifest, it loads the extension from your local web server instance. For example, IISExpress in Visual Studio. After you change the manifest, deploy and install this debugging extension only once.

> **Note**
>
> Run your local web server in SSL mode because Azure DevOps demands that the web page is served from a secure source. Otherwise, you get an error in the browser console during the extension IFRAME loading.

Update your extension

To change an extension that's already published, update it.

> **Tip**
>
> We recommend updating the extension over removing and re-uploading. We also recommend having two extensions, for example, `publisher.extension` and `publisher.extension-dev`. `Publisher.extension` is public in the Marketplace, where customers can install it in their Azure DevOps organizations. `Publisher.extension-dev` is kept private in the Marketplace and can be shared with an organization that you own and control. You don't need to maintain two copies of source code of the extension. You can maintain two manifest files - one for each extension and during packaging of the extension you can provide the respective manifest file to the tfx-cli tool. For more information on arguments required for the tool, see **TFX extension commands**.

1. Select an extension from the list of displayed items.
2. Right-click and select **Update** for the `publisher.extension-dev`, for example.
3. Validate your extension.
4. Make the same updates to the production version, `publisher.extension`, for example.
5. Browse to the .vsix for your extension and upload it.

The updated version of your extension automatically gets installed to accounts that have it already installed. New accounts where the extension is installed in the future also receive the latest version.

Make your extension public

While you develop your extension or integration for the Marketplace, keep it private. To make your extension available publicly, set the public flag to `true` in your manifest.

Qualifications

To have a public listing on the Marketplace, your integration or extension must meet the following qualifications:

- Works with or extends Azure DevOps.
- You, or your company, own, develop, and are licensed to distribute and advertise the integration or extension.
- The extension or integration is actively maintained.

Microsoft might also request a demo and to review the content planned for your Marketplace entry.

Top Publisher

The Top Publisher program is only available for publishers with Azure DevOps extensions or integrations. It's not applicable for Visual Studio IDE and Visual Studio Code extension publishers.

The Top Publisher program recognizes publishers with commitment to their customers and the Marketplace through exemplary policies, quality, reliability, and support. Once you become a Top Publisher, all of your public offerings display the Top Publisher badge.

Top Publisher requirements

The Top Publisher program in the Marketplace is designed to help you evaluate or acquire Azure DevOps extensions and integrations with confidence. The Top Publisher badge implies that the publisher shows commitment to their customers and the Marketplace through exemplary policies, quality, reliability, and support. It's for

publishers with one or more global Azure DevOps extensions or integrations and isn't applicable for Visual Studio IDE and Visual Studio Code extension publishers.

Marketplace assigns the badge to a publisher after carefully reviewing the publisher across the following parameters:

- Privacy policy
- Licensing policy
- Support policy
- Documentation
- Q & A responsiveness
- Ratings & review for their offerings
- Active uptake and the install count for their offerings
- Manage at least one public Azure DevOps extension from the publisher.
- The public extension should have more than 5000 installs, with an active install count exceeding 1000.

You can expect timely support and a good overall experience when you get an extension from a Top Publisher. Check out the offerings from the Top Publishers.

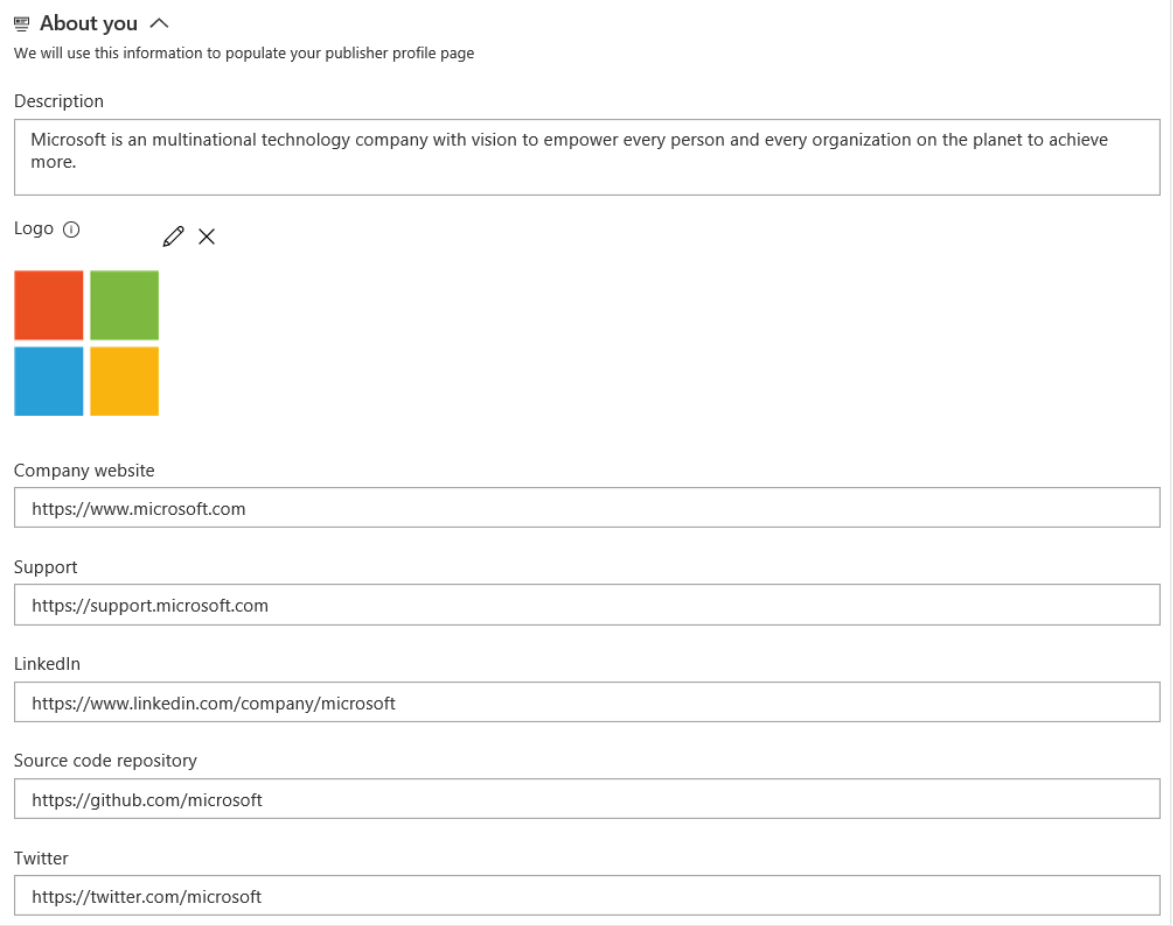
For more information on adding policies to your offering, see the extension manifest.

1. Update your publisher profile.

   Through the publisher profile, you can showcase all of your offerings in one place along with key publisher-related information. To provide the information, which shows up in the profile, do the following steps:

   a. Sign in to https://marketplace.visualstudio.com/manage/publishers using the account with which you publish and manage your offerings in the Visual Studio Marketplace.

b. Select the publisher and complete the **About you** section in the **Details** tab.



c. Save your changes and select **View profile** to see how it appears to consumers. You can use this profile page to evangelize your offering(s).

> **Note**
>
> Through this program, it is the publisher that is being certified. This doesn't cover the software or security of their extensions and integrations. We recommend you be aware of the safety information when you're evaluating the offerings from a publisher.

If you got an extension from a Top Publisher and aren't satisfied with your experience, consider engaging with the publisher first.

Apply to be a Top Publisher

1. Sign in to https://marketplace.visualstudio.com/manage/publishers using the account with which you publish and manage your offerings in Marketplace
2. Select the publisher and navigate to its **Top Publisher** tab. Note: you need to have one or more global Azure DevOps (Server/Service) extension or integration for the tab to appear.

3. If you meet part of the previously listed requirements and are the publisher's owner, you see an option to apply for the program. On application, an email is sent to the Marketplace team to review your case. They respond in under 10 business days with next steps, clarifying questions or with the grant of the badge.

The team likely looks at other parameters, such as active uptake of your offerings, install/get started counts and ratings & reviews across your offerings before granting the badge. Microsoft reserves the right to grant, reject or revoke the Top Publisher badge at any time.

Once a publisher is a Top Publisher, then all its future updates and offerings must meet the previously listed requirements.

Respond to Marketplace extension reviews

You can respond to reviews that customers leave for your extensions in the Visual Studio Marketplace. Find and select **Reply** next to a review if you have one of the following permissions: owner, creator, or contributor.

You can leave only one response. Avoid using reviews as a support forum. If you need more details, please provide a support alias for the reviewer to contact. You can then resolve their problems externally and update your reply with a resolution.

Guidelines for publisher responses

Keep the Visual Studio Marketplace an open, inviting, respectful, and helpful place for customers to find, try, install, and review extensions. Communication plays an important role in keeping a healthy community. To help create this environment, here are guidelines for publishers responding to customer reviews. Think deeply about your customer interactions and reflect on the spirit of the customer experience that the Marketplace is trying to create.

- Reviews are reserved for customer comments. Use *Reply* only to respond to a review.
- Reviews are for sharing customer opinions, so *all opinions are valid*. Customers are entitled to their opinions, so treat comments respectfully as feedback without debate, criticism, or argument.
- Make sure that your responses add value and are relevant to your customers' comments.
- Focus on precisely addressing questions or problems. If you need more details, ask the customer to contact you over email, rather discuss in reviews. When you

resolve the problem, update your reply with the resolution. You can edit your reply, just like customers can edit their reviews.
- If you come across any inappropriate reviews, like spam, abusive, or offensive content, for any extension, flag it for our review.

Request to void a review

As a publisher, you can appeal to void a review if the issue reported is because of the Marketplace or underlying platform. If the issue is valid, Marketplace admins void the rating. You can **Appeal** from ratings and review section on your extension hub page.

Unpublish an extension

You can unpublish free extensions, if you no longer want to offer them in the Marketplace.

The following scenarios cover when you might want to remove your extension from the Marketplace:

- You developed another extension and no longer want to offer the current one.
- Your extension has a problem, so you want to remove your extension from the Marketplace until you resolve the problem.
- You published your extension as public by mistake.

Certain criteria must be met for an extension to be unpublished or removed:

Expand table

Action	Requirements
Unpublish	Only **free extensions** might be unpublished.
Remove	Your extension must have **zero (0)** installs to be removed.

> **Important**
>
> If you must remove your extension because of legal or security problems, contact Customer Support at the Developer Community. We review the request and manually delete the extension.

1. Select the extension on your publisher page and choose **Unpublish** on the menu.

Your extension is unpublished immediately from the Marketplace, and new users can't install it. Ratings and reviews for your extension stay intact.

To offer your extension again in the Marketplace, choose **Publish** on the menu.

You can also choose to remove your extension completely from the Marketplace if your extension has zero (0) installs. To do so, choose **Remove** on the menu. This action can't be undone.

Related articles

- Develop a web extension
- Extensibility points

Feedback

Was this page helpful? Yes No

Provide product feedback

Publish from the command line

Article • 12/26/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

You can use the Cross-platform CLI for Azure DevOps (tfx-cli) to publish your extension to the Visual Studio Marketplace.

For more information, see the overview of publish, install, and share.

Prerequisites

Get the TFX CLI from Node Package Manager and generate a Microsoft Entra token or a personal access token (PAT). Also, if you haven't already, set up a Publisher in the Gallery.

Acquire the Cross-platform CLI for Azure DevOps

1. If you don't have it, download and install NodeJS. During set up, ensure that you leave **Add to PATH** chosen.
2. Open a Command Prompt and enter `npm i -g tfx-cli`.

If you already have the TFX CLI installed, you can update to the latest release by running `npm up -g tfx-cli`.

Publish with a Microsoft Entra token as a service principal

It is also possible to publish an extension as a service principal.

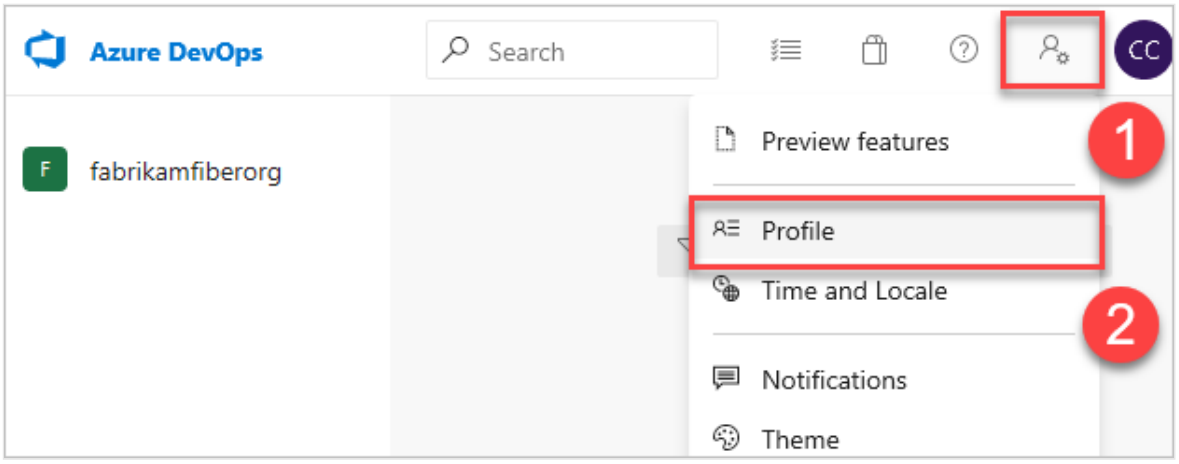
1. Add the service principal as a member to a publisher account. You can get the service principal's ID from its profile using Profiles - Get. Then, you can add the service principal as a member to the publisher using the ID from the previous step.
2. Publish an extension via TFX CLI using a service principal. Execute the following TFX CLI command to use its access token:

```
tfx extension publish --publisher my-publisher --vsix my-publisher.my-
extension-1.0.0.vsix --auth-type pat -t <ENTRA_TOKEN>
```

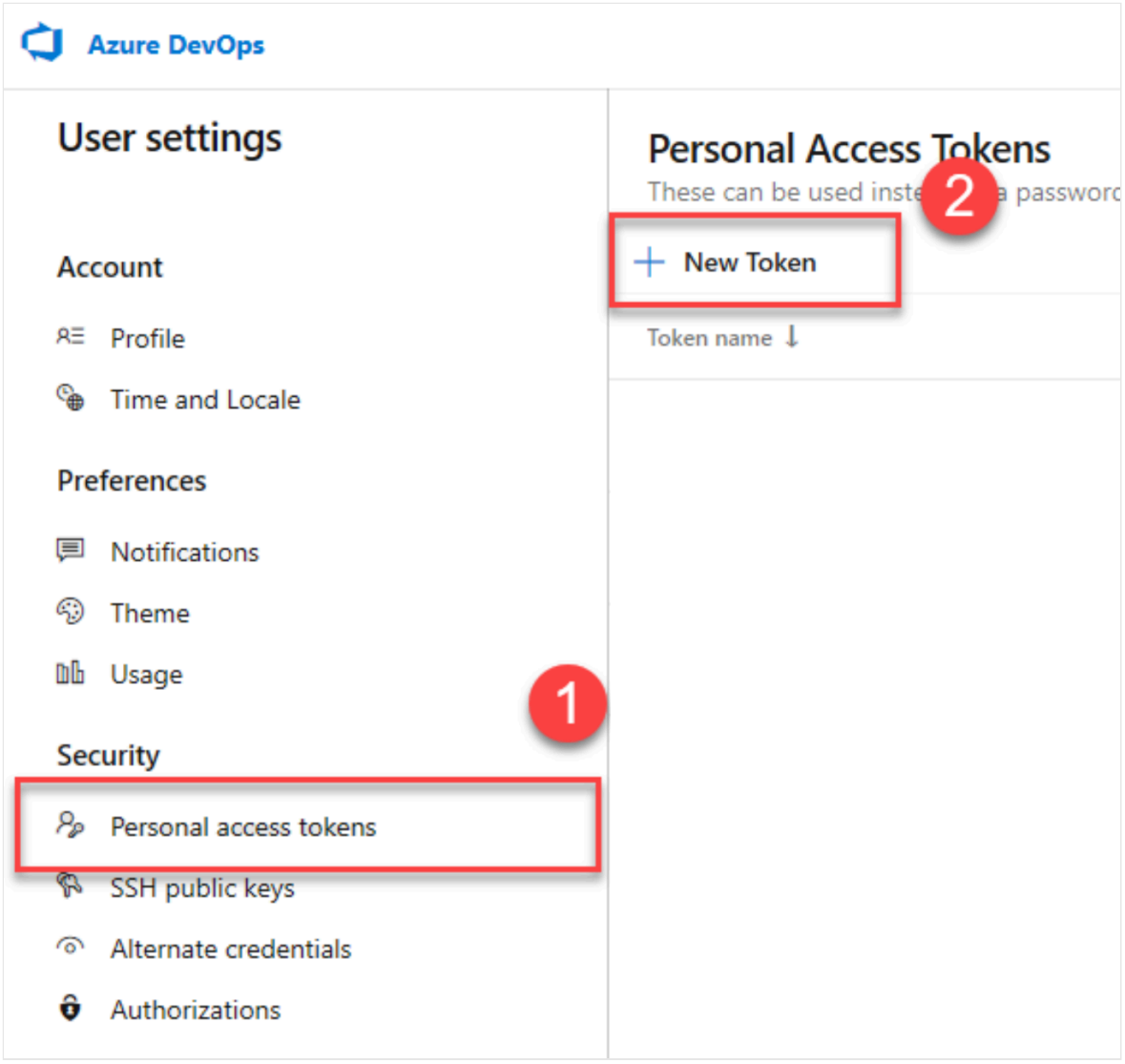
Publish with a personal access token

1. Sign in to your organization (`https:dev.azure.com/{organization}`).

2. From your home page, open the user settings icon, and then select **Profile**.

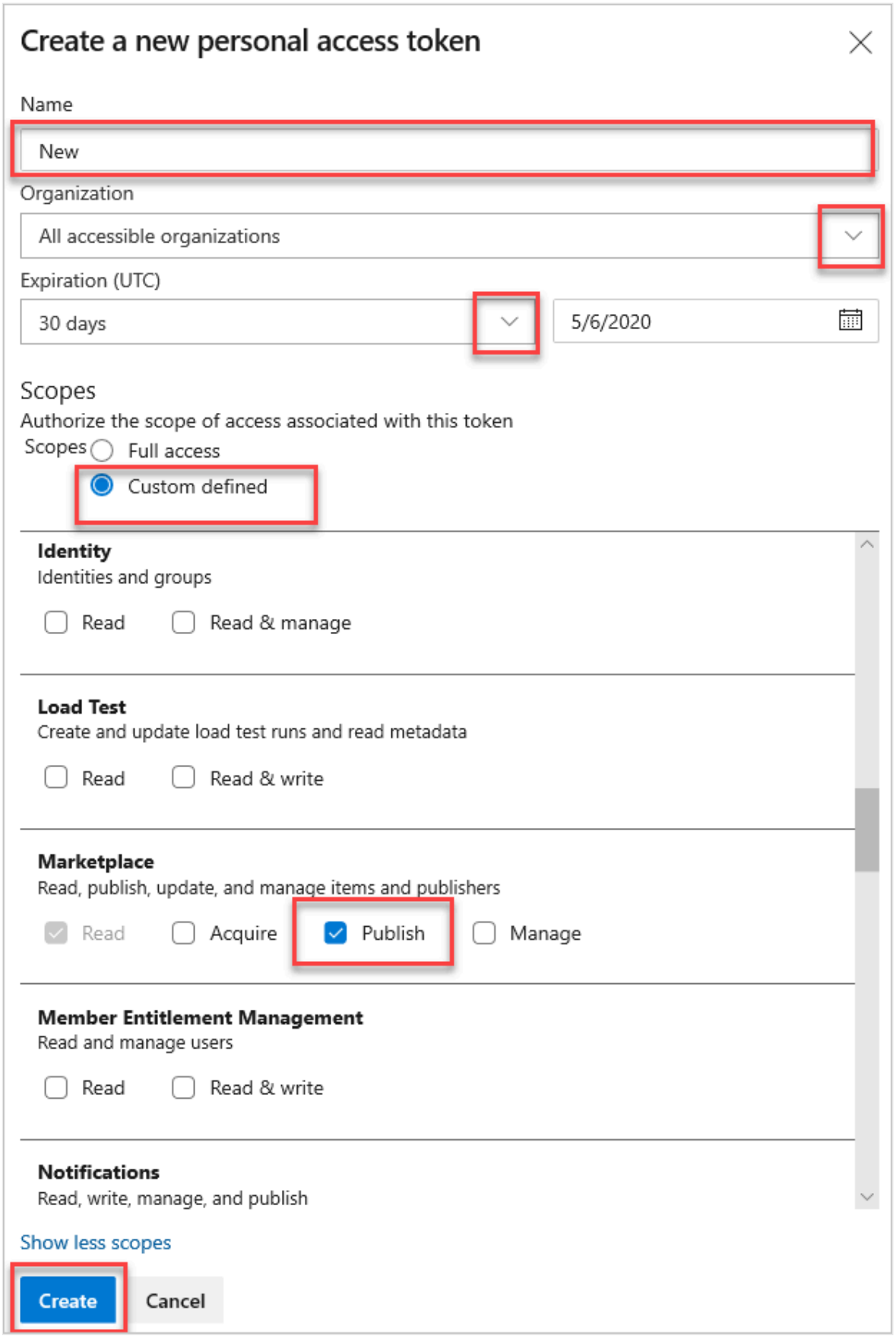


3. Under **Security**, select **Personal access tokens**, and then select **New Token**.

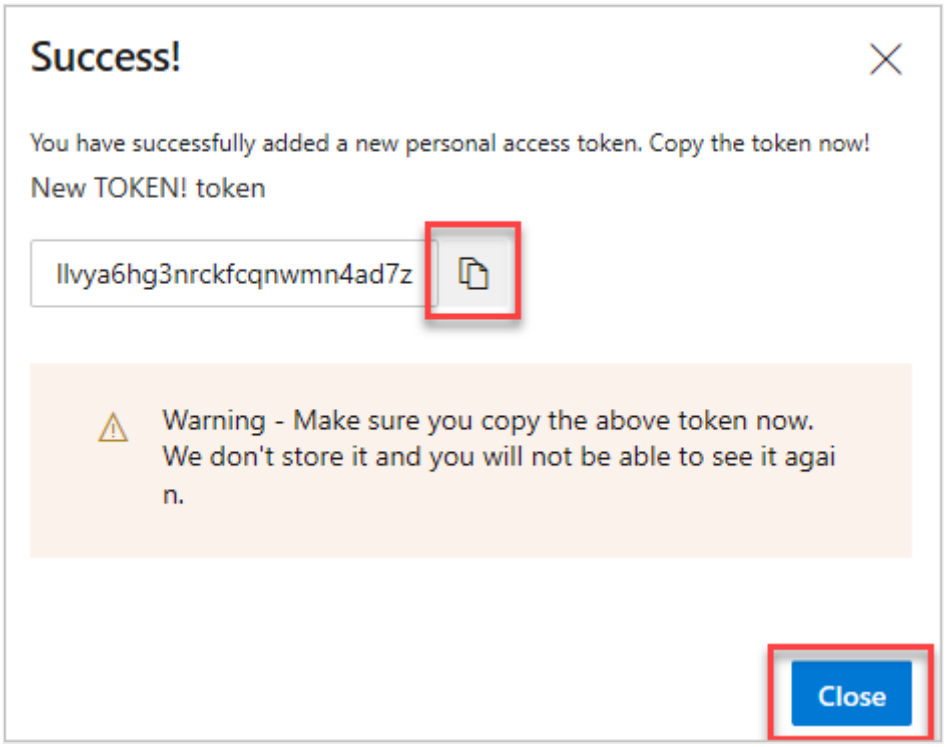


4. Complete the form:

- Name your token
- Select **All accessible organizations**, which is the only value that works when publishing via CLI. If you select only one organization, it results in an error, even if the PAT is valid
- Select an expiration time frame for your token. This is required because the Visual Studio Marketplace publishing APIs work outside of the context of an organization
- Set the scope of access associated with this token. Be sure to select the **Marketplace (publish)** scope. This scope limits the token to only being able to publish extensions to the Marketplace.
- Select **Create**



5. Copy your generated personal access token. Make sure to keep it secret.



Once TFX CLI is installed and you have your token, you can use the tool to package and publish your extension.

1. Open a command prompt to the root directory of your extension.
2. Run the following command to publish your extension. When prompted, enter your token to authenticate.

Command

```
tfx extension publish --publisher <YOUR_PUBLISHER_ID> --manifest-js
<YOUR_EXTENSION_MANIFEST> --share-with <ACCOUNT_NAME>
```

Potential Errors

You may receive the following error if your extension has already been published:

```
Failed Request: Internal Server Error(500) - Version number must increase
each time an extension is published. Extension: fabrikam.my-extension
Current version: 0.1.9  Updated version: 0.1.9
```

You can add the `--rev-version` flag to automatically increment the *patch* version of your extension. This also saves the new version to your manifest.

> ⓘ **Note**

All options available for `create` are available for the `publish` command.

Example

```
C:\vso-team-calendar>tfx extension publish --publisher publishFabrikam --
manifest-js fabrikam.config.js --share-with fabrikam --rev-version
Copyright Microsoft Corporation
> Personal access token:
Checking if this extension is already published
It is, update the extension
Waiting for server to validate extension package...
Sharing extension with fabrikam.

=== Completed operation: publish extension ===
 - Packaging: C:\vso-team-calendar\fabrikam.team-calendar-0.2.6.vsix
 - Publishing: success
 - Sharing: shared with fabrikam
```

Feedback

Was this page helpful? Yes No

Provide product feedback

Package and publish extensions

Article • 01/09/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Once you develop your extension, you can package and publish it to the Visual Studio Marketplace. The Marketplace is a global repository for private and public extensions, integrations, and other offers from Microsoft.

> **Note**
>
> For information on the discovery properties available in your extension's manifest file that helps users discover and learn about your extension, see the Extension Manifest Reference.

Prerequisites

The following list of requirements must be met before you publish to the Marketplace.

- Install the extension packaging tool (TFX). Run `npm install -g tfx-cli` from a command prompt.
- Ensure the proper permissions are granted to use any images, for example, icons, logos, screenshots, and so on.
- Include a thorough `overview.md` file to describe your listing in the Marketplace.
- Include an icon for your extension, which is at least 128x128 pixels in size.
- When you refer to Microsoft products, use full names in place of abbreviations, for example, Azure DevOps vs. AzDO or - any other abbreviation.
- Refrain from using brand names in the name of your extension.

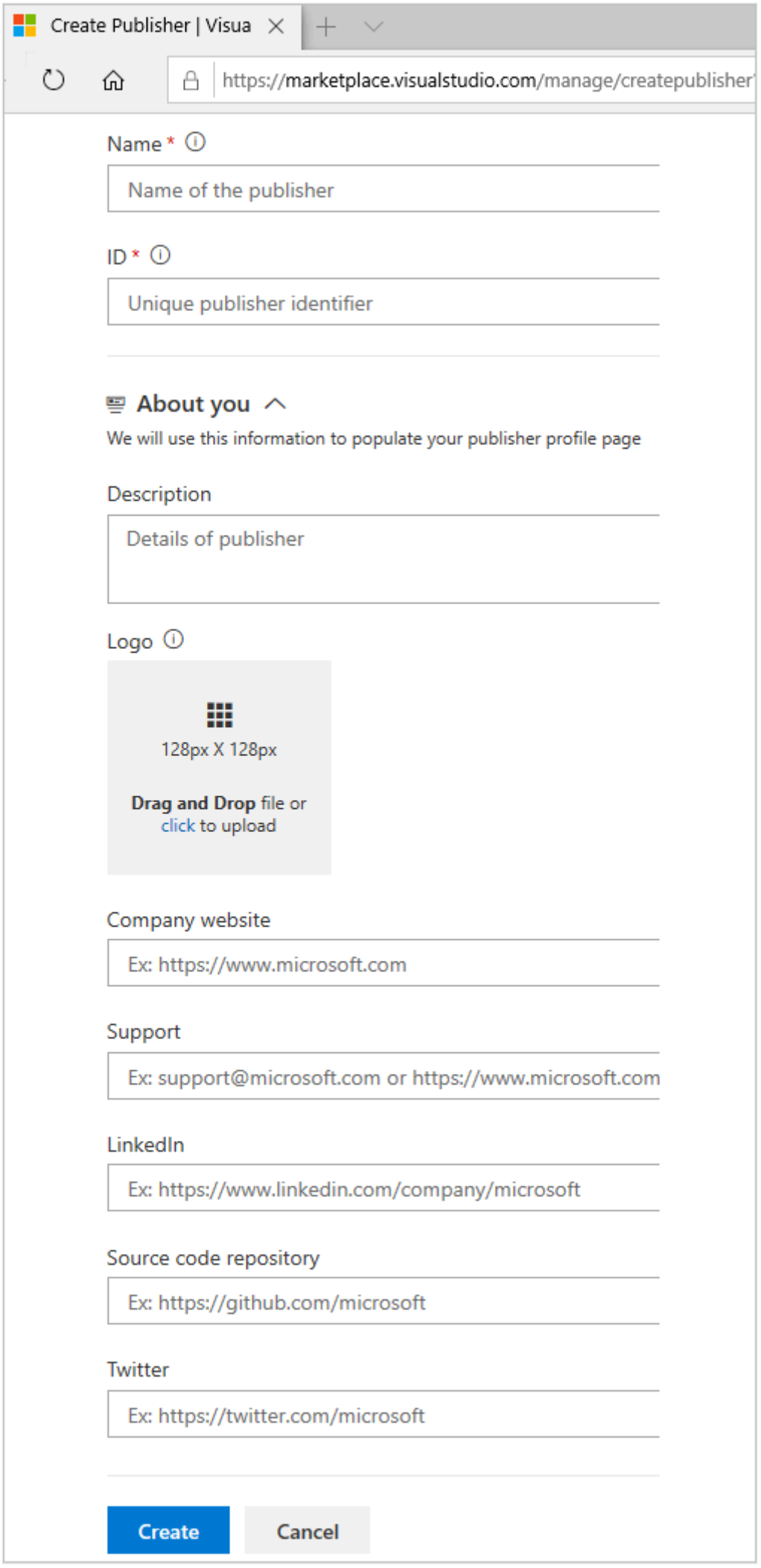
Create a publisher

All extensions and integrations, including extensions from Microsoft, have a publisher. Anyone can create a publisher and publish extensions under it. You can also give other people access to your publisher if a team is developing the extension.

A user owns the publisher, typically the user who created it. You can also share the publisher with other users.

1. Sign in to the Visual Studio Marketplace Publishing Portal.

2. If you're not already a member of an existing publisher, **+ Create a publisher**. Enter a name in the publisher name field. The ID field should automatically get set based on the name you entered.

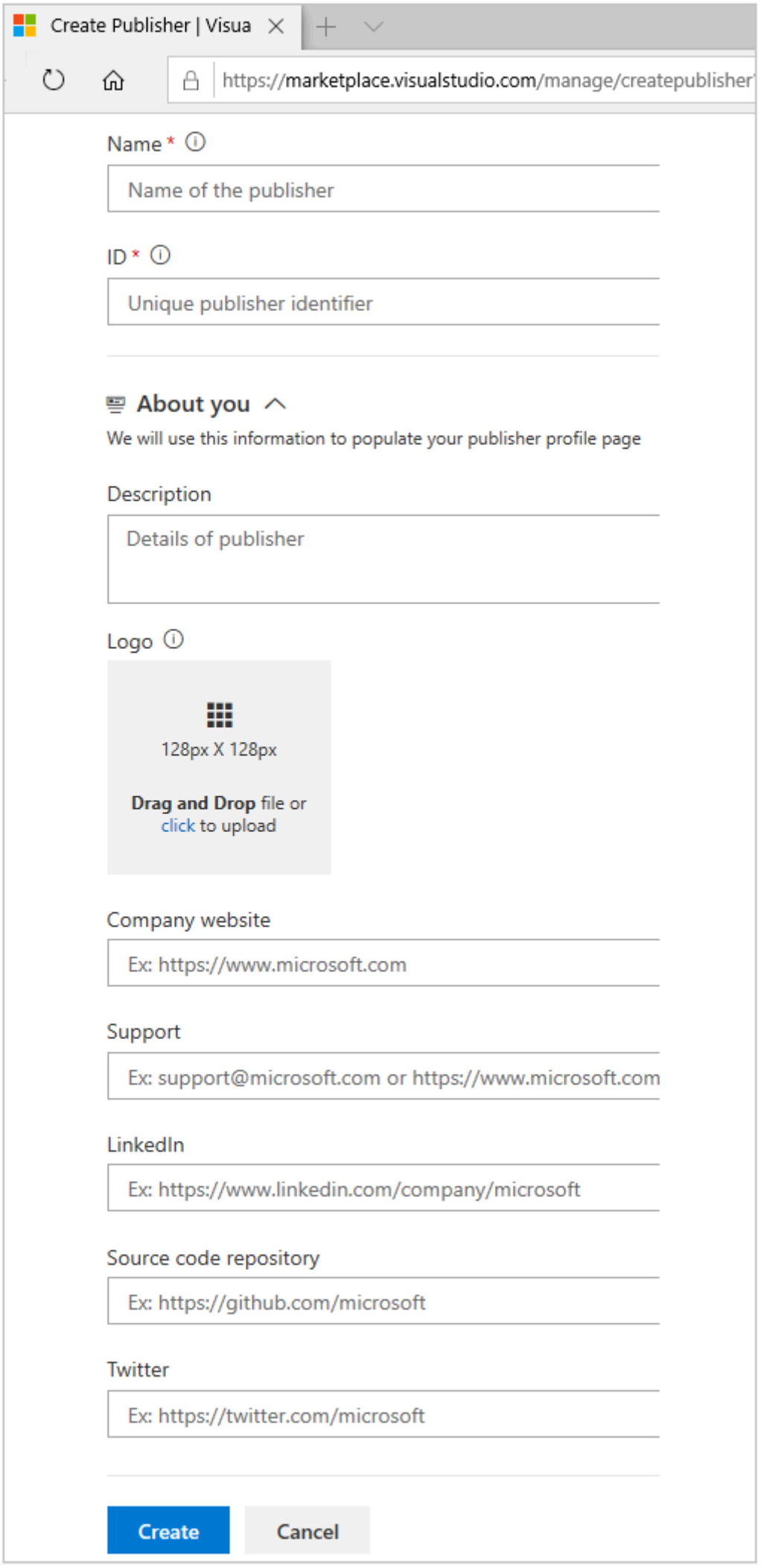

Create Publisher | Visua
https://marketplace.visualstudio.com/manage/createpublisher
Name *
Name of the publisher
ID *
Unique publisher identifier
About you
We will use this information to populate your publisher profile page
Description
Details of publisher
Logo
128px X 128px
Drag and Drop file or
click to upload
Company website
Ex: https://www.microsoft.com
Support
Ex: support@microsoft.com or https://www.microsoft.com
LinkedIn
Ex: https://www.linkedin.com/company/microsoft
Source code repository
Ex: https://github.com/microsoft
Twitter
Ex: https://twitter.com/microsoft
Create
Cancel

> **Note**
>
> Make note of the ID, as you need to set it in the manifest file of your extension.

If you're not prompted to create a publisher, scroll down to the bottom of the page and select **Publish extensions** below *Related sites*.

- Specify an identifier for your publisher, for example: `mycompany-myteam`. This identifier is used as the value for the `publisher` attribute in your extension manifest file.
- Specify a display name for your publisher, for example: `My Team`

3. Review the Marketplace Publisher Agreement, and then select **Create**.


Create Publisher | Visua
https://marketplace.visualstudio.com/manage/createpublisher
Name *
Name of the publisher
ID *
Unique publisher identifier
About you
We will use this information to populate your publisher profile page
Description
Details of publisher
Logo
128px X 128px
Drag and Drop file or
click to upload
Company website
Ex: https://www.microsoft.com
Support
Ex: support@microsoft.com or https://www.microsoft.com
LinkedIn
Ex: https://www.linkedin.com/company/microsoft
Source code repository
Ex: https://github.com/microsoft
Twitter
Ex: https://twitter.com/microsoft
Create
Cancel

Once the publisher's created, you're directed to manage items, but there aren't any items.

Package your extension

To upload your extension, you need to package it as a VSIX 2.0-compatible .vsix file. Microsoft provides a cross-platform command-line interface (CLI) to package and publish your extension.

1. Open your extension manifest file (`vss-extension.json`) and set the value of the `publisher` field to the ID of your publisher. For example:

   JSON

   ```
   {
       ...
       "id": "my-first-extension",
       "publisher": "AnnetteNielsen",
       ...
   }
   ```

2. From a command prompt, run the TFX tool's packaging command from your extension directory.

   ```
   npx tfx-cli extension create
   ```

   A message displays indicating your extension is successfully packaged:

   ```
   === Completed operation: create extension ===
   - VSIX: C:\my-first-extension\AnnetteNielsen.my-first-extension-
   1.0.0.vsix
   - Extension ID: my-first-extension
   - Extension Version: 1.0.0
   - Publisher: AnnetteNielsen
   ```

> **Note**
>
> An extension/integration's version must be incremented on every update.
> If you haven't incremented your extension/integration in the manifest, you should

pass the `--rev-version` command line switch. This increments the *patch* version number of your extension and saves the new version to your manifest.

Check package size

Check the size of the vsix after it gets packaged. If it's greater than 50 MB, you need to optimize it. To do so, see the following considerations:

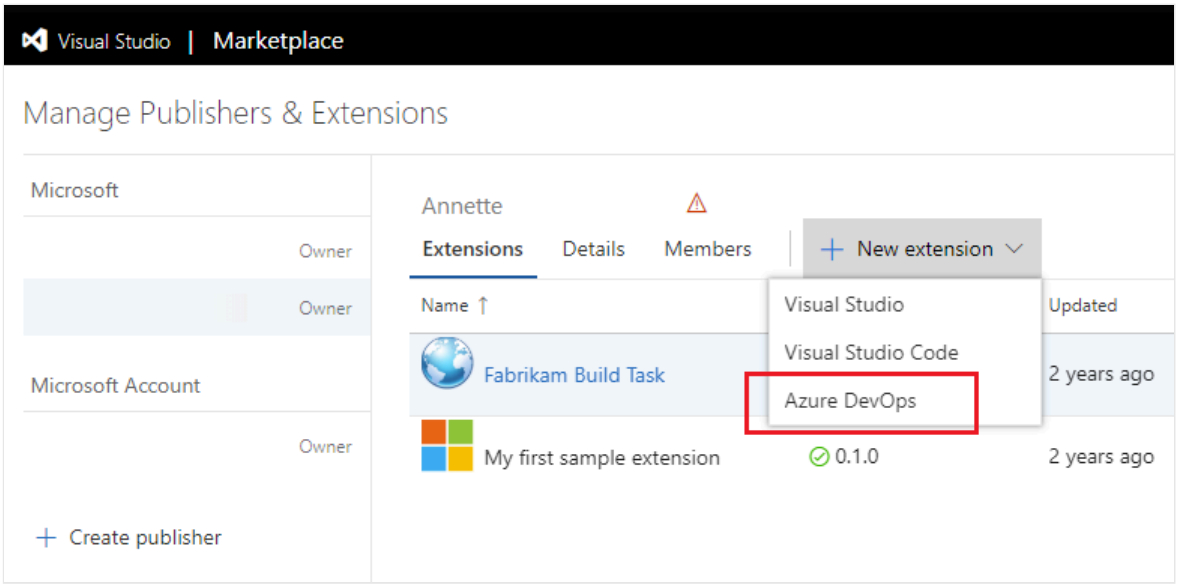
- Deduplicate the common dependencies, if any, by stating them once in the extension package.
- Fetch things at runtime or during install time rather than providing it within the package. Consider using the tool installer lib to pull tool dependencies at runtime. Using the lib offers benefits where the tool gets cached by version so for private agents, it doesn't download every build. We made it a lib so it can be used outside of tool installer tasks. But, the task doesn't work in disconnected scenarios (no internet), which should be in the description / docs for the task.
- Some users have success with WebPack to tree shake their dependencies in their tasks.

Publish your extension

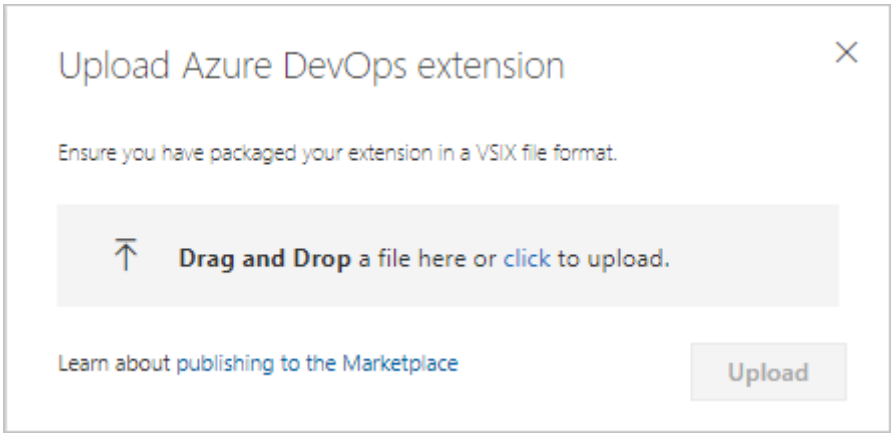
Once your extension is packaged, you can upload it to the Marketplace under a publisher. The `publisher` identifier specified in your extension's manifest file must match the identifier of the publisher the extension is uploaded under.

1. From the management portal, select your publisher from the drop-down menu at the top of the page.

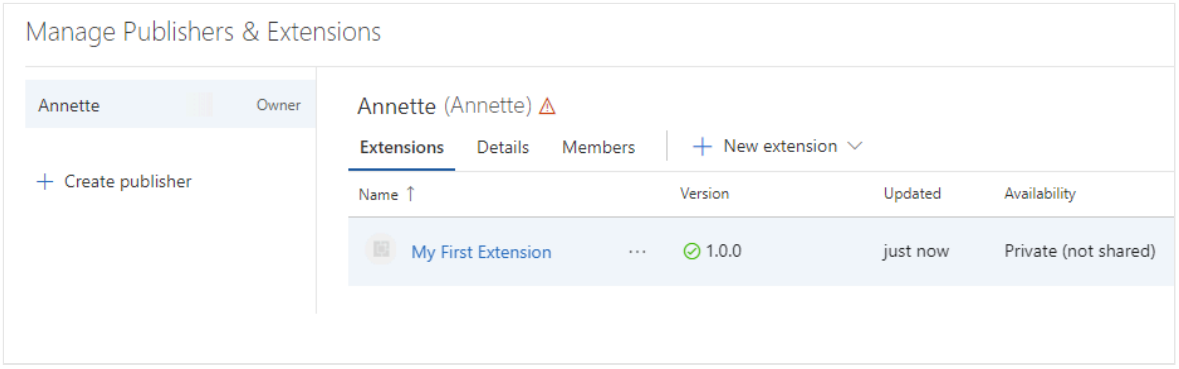
2. Select **New extension** > **Azure DevOps**.



3. Drag and drop your file or select it to find your VSIX file, which you created in the previous packaging step, and then choose **Upload**.



After quick validation, your extension appears in the list of published extensions. Don't worry, the extension is only visible to you.



At this point, your extension isn't visible to any accounts and can't be installed until you share it.

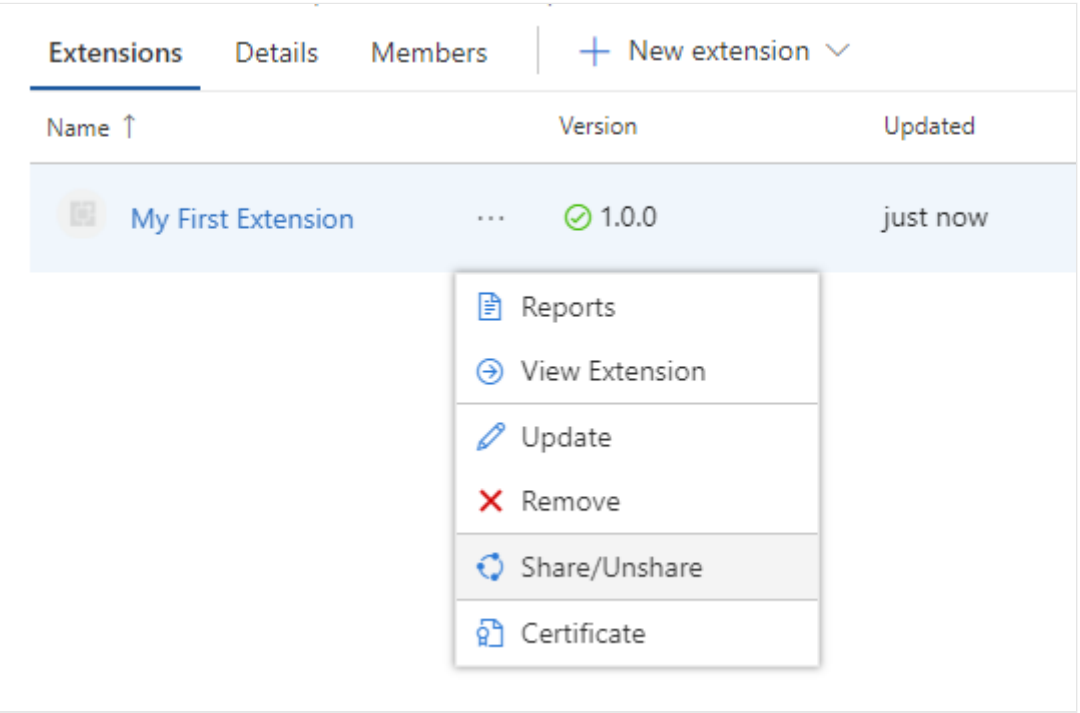
> **Note**

Microsoft runs a virus scan on each new and updated extension package published. Until the scan is all clear, we don't publish the extension in the Marketplace for public usage. This way we also avoid surfacing inappropriate or offensive content on the Marketplace pages.

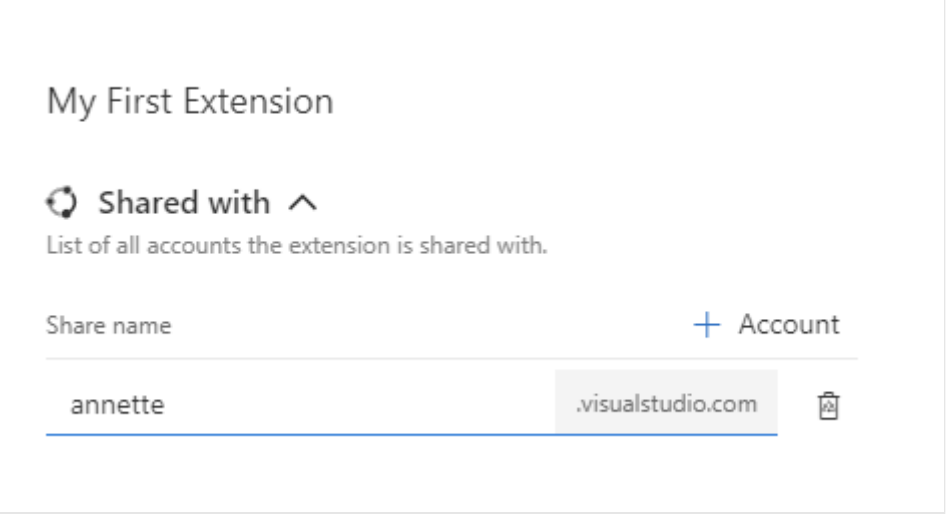
Share your extension

You must share your extension with an organization before you can install it in Azure DevOps. To share an extension, do the following tasks:

1. From the Marketplace management portal, select your extension from the list, right-click, and then choose **Share/Unshare** or **Publish/Unpublish**, depending on the extension.



2. Select **Organization**, and then enter the name of your organization. Select **Enter**.



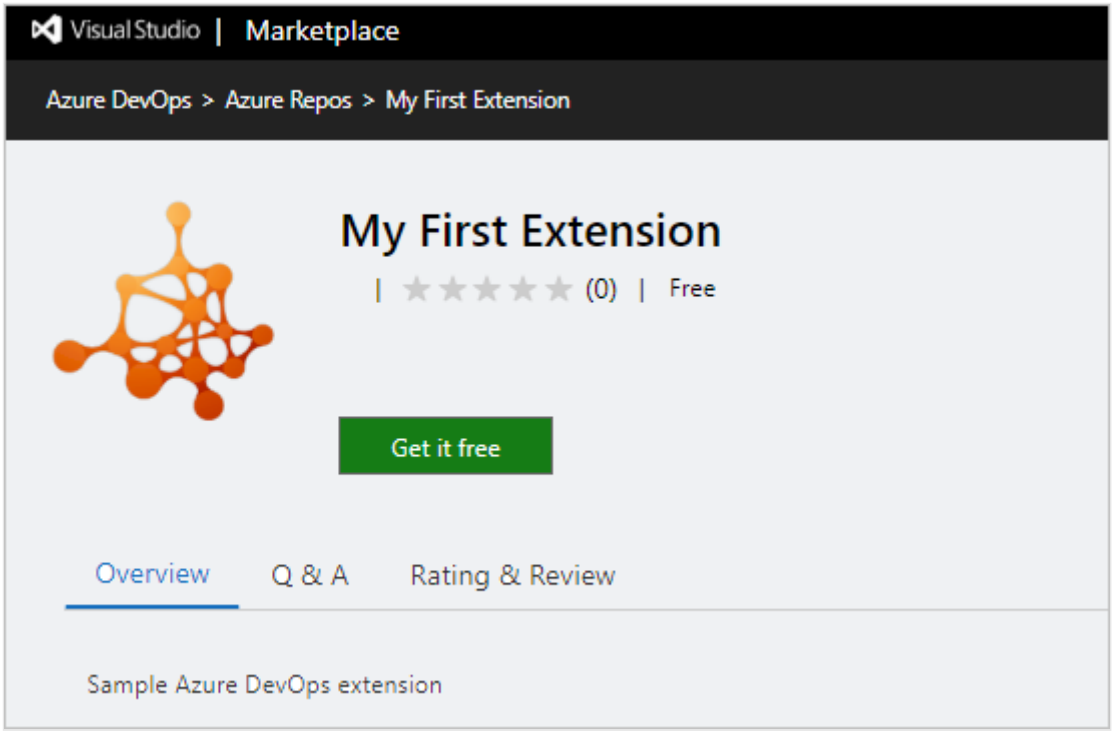
3. Close the panel.

Your extension can now be installed into this organization.

Install your extension

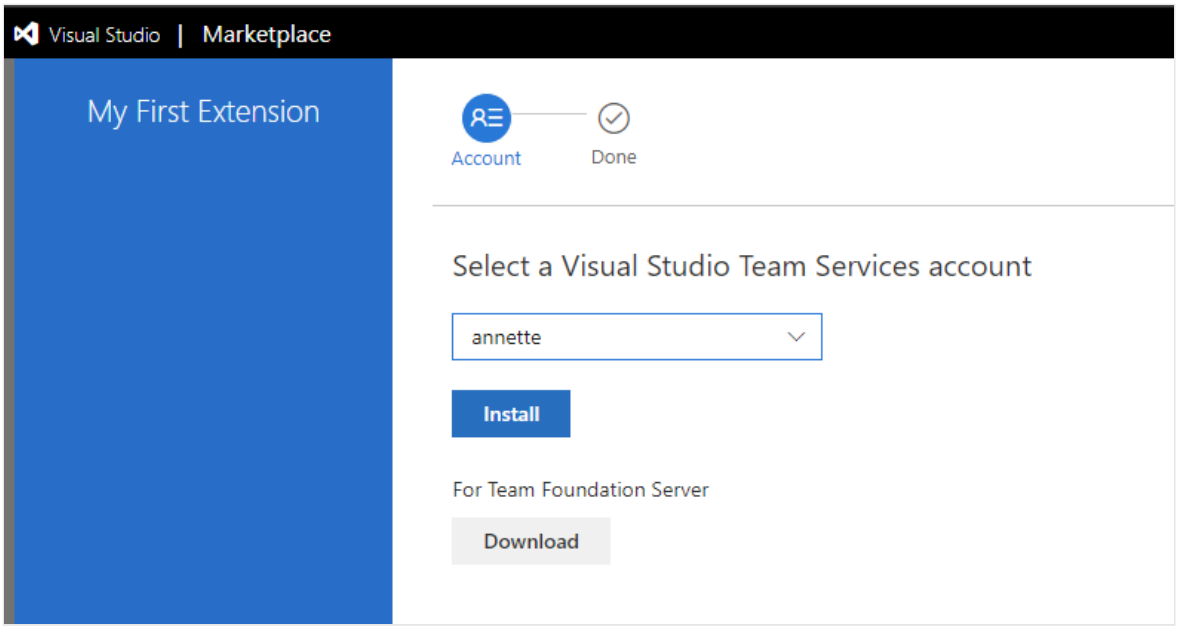
To install your shared extension, do the following steps.

1. In the Marketplace, select your extension to open its overview page.



> ⓘ **Note**
>
> Because your extension is private, only you and any member of the organization it's shared with can see this page.

2. Select **Get it free** to start the installation process. Select the organization you shared the extension with from the dropdown menu.



3. Select **Install**.

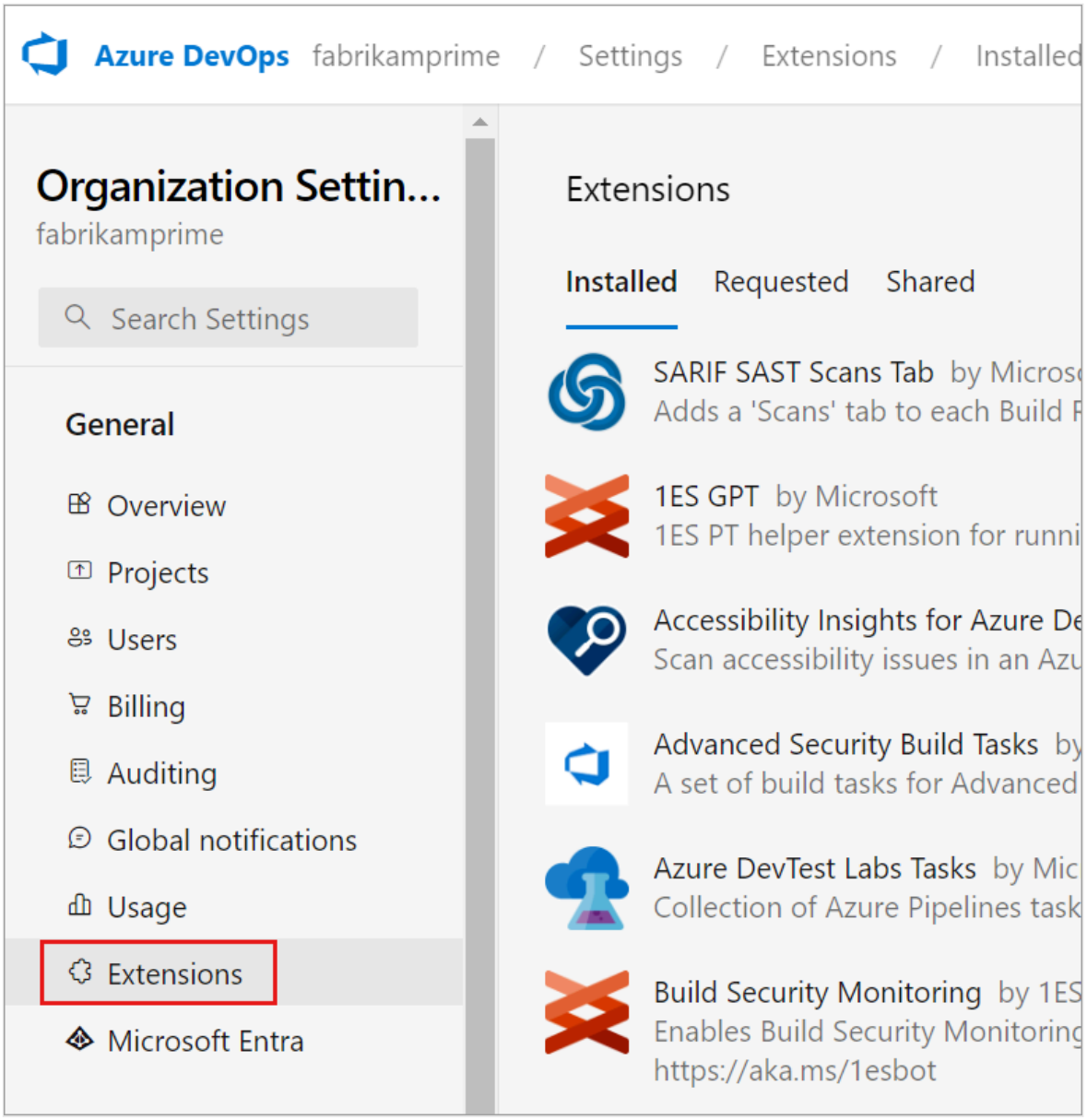
Congratulations! You installed your extension into an organization and you're ready to try it.

Try your extension

1. Select **Proceed to organization** at the end of the installation wizard to go to the home page of the organization the extension was installed to (`https://dev.azure.com/{organization}`).

2. Refresh your browser.

3. Open **Organization settings**, and then select **Extensions**.



You should see the new extension on the **Installed** tab.

Debug your extension

To debug the extension using Visual Studio or Browser Developer Tools, change the manifest by adding the `baseUri` property. This action speeds up the development without the need to redeploy the extension each time you change source code.

JSON

```
{
    ...
    "baseUri": "https://localhost:44300",
    ...
}
```

When you change the manifest, it loads the extension from your local web server instance. For example, IISExpress in Visual Studio. After you change the manifest, deploy and install this debugging extension only once.

> **Note**
>
> Run your local web server in SSL mode because Azure DevOps demands that the web page is served from a secure source. Otherwise, you get an error in the browser console during the extension IFRAME loading.

Update your extension

To change an extension that's already published, update it.

> **Tip**
>
> We recommend updating the extension over removing and re-uploading. We also recommend having two extensions, for example, `publisher.extension` and `publisher.extension-dev`. `Publisher.extension` is public in the Marketplace, where customers can install it in their Azure DevOps organizations. `Publisher.extension-dev` is kept private in the Marketplace and can be shared with an organization that you own and control. You don't need to maintain two copies of source code of the extension. You can maintain two manifest files - one for each extension and during packaging of the extension you can provide the respective manifest file to the tfx-cli tool. For more information on arguments required for the tool, see **TFX extension commands**.

1. Select an extension from the list of displayed items.
2. Right-click and select **Update** for the `publisher.extension-dev`, for example.
3. Validate your extension.
4. Make the same updates to the production version, `publisher.extension`, for example.
5. Browse to the .vsix for your extension and upload it.

The updated version of your extension automatically gets installed to accounts that have it already installed. New accounts where the extension is installed in the future also receive the latest version.

Make your extension public

While you develop your extension or integration for the Marketplace, keep it private. To make your extension available publicly, set the public flag to `true` in your manifest.

Qualifications

To have a public listing on the Marketplace, your integration or extension must meet the following qualifications:

- Works with or extends Azure DevOps.
- You, or your company, own, develop, and are licensed to distribute and advertise the integration or extension.
- The extension or integration is actively maintained.

Microsoft might also request a demo and to review the content planned for your Marketplace entry.

Top Publisher

The Top Publisher program is only available for publishers with Azure DevOps extensions or integrations. It's not applicable for Visual Studio IDE and Visual Studio Code extension publishers.

The Top Publisher program recognizes publishers with commitment to their customers and the Marketplace through exemplary policies, quality, reliability, and support. Once you become a Top Publisher, all of your public offerings display the Top Publisher badge.

Top Publisher requirements

The Top Publisher program in the Marketplace is designed to help you evaluate or acquire Azure DevOps extensions and integrations with confidence. The Top Publisher badge implies that the publisher shows commitment to their customers and the Marketplace through exemplary policies, quality, reliability, and support. It's for

publishers with one or more global Azure DevOps extensions or integrations and isn't applicable for Visual Studio IDE and Visual Studio Code extension publishers.

Marketplace assigns the badge to a publisher after carefully reviewing the publisher across the following parameters:

- Privacy policy
- Licensing policy
- Support policy
- Documentation
- Q & A responsiveness
- Ratings & review for their offerings
- Active uptake and the install count for their offerings
- Manage at least one public Azure DevOps extension from the publisher.
- The public extension should have more than 5000 installs, with an active install count exceeding 1000.

You can expect timely support and a good overall experience when you get an extension from a Top Publisher. Check out the offerings from the Top Publishers.

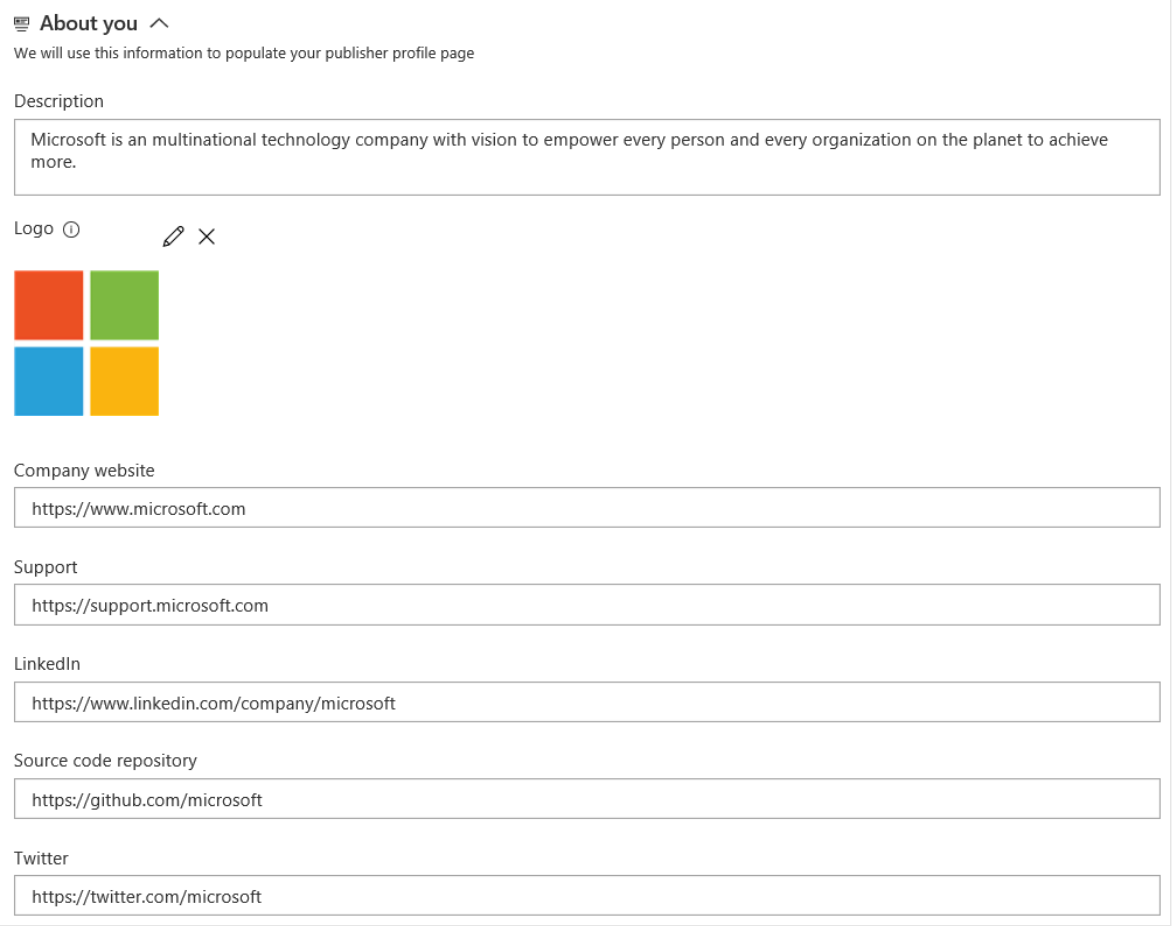
For more information on adding policies to your offering, see the extension manifest.

1. Update your publisher profile.

   Through the publisher profile, you can showcase all of your offerings in one place along with key publisher-related information. To provide the information, which shows up in the profile, do the following steps:

   a. Sign in to https://marketplace.visualstudio.com/manage/publishers using the account with which you publish and manage your offerings in the Visual Studio Marketplace.

b. Select the publisher and complete the **About you** section in the **Details** tab.



c. Save your changes and select **View profile** to see how it appears to consumers. You can use this profile page to evangelize your offering(s).

> **Note**
>
> Through this program, it is the publisher that is being certified. This doesn't cover the software or security of their extensions and integrations. We recommend you be aware of the safety information when you're evaluating the offerings from a publisher.

If you got an extension from a Top Publisher and aren't satisfied with your experience, consider engaging with the publisher first.

Apply to be a Top Publisher

1. Sign in to https://marketplace.visualstudio.com/manage/publishers using the account with which you publish and manage your offerings in Marketplace
2. Select the publisher and navigate to its **Top Publisher** tab. Note: you need to have one or more global Azure DevOps (Server/Service) extension or integration for the tab to appear.

3. If you meet part of the previously listed requirements and are the publisher's owner, you see an option to apply for the program. On application, an email is sent to the Marketplace team to review your case. They respond in under 10 business days with next steps, clarifying questions or with the grant of the badge.

The team likely looks at other parameters, such as active uptake of your offerings, install/get started counts and ratings & reviews across your offerings before granting the badge. Microsoft reserves the right to grant, reject or revoke the Top Publisher badge at any time.

Once a publisher is a Top Publisher, then all its future updates and offerings must meet the previously listed requirements.

Respond to Marketplace extension reviews

You can respond to reviews that customers leave for your extensions in the Visual Studio Marketplace. Find and select **Reply** next to a review if you have one of the following permissions: owner, creator, or contributor.

You can leave only one response. Avoid using reviews as a support forum. If you need more details, please provide a support alias for the reviewer to contact. You can then resolve their problems externally and update your reply with a resolution.

Guidelines for publisher responses

Keep the Visual Studio Marketplace an open, inviting, respectful, and helpful place for customers to find, try, install, and review extensions. Communication plays an important role in keeping a healthy community. To help create this environment, here are guidelines for publishers responding to customer reviews. Think deeply about your customer interactions and reflect on the spirit of the customer experience that the Marketplace is trying to create.

- Reviews are reserved for customer comments. Use *Reply* only to respond to a review.
- Reviews are for sharing customer opinions, so *all opinions are valid*. Customers are entitled to their opinions, so treat comments respectfully as feedback without debate, criticism, or argument.
- Make sure that your responses add value and are relevant to your customers' comments.
- Focus on precisely addressing questions or problems. If you need more details, ask the customer to contact you over email, rather discuss in reviews. When you

resolve the problem, update your reply with the resolution. You can edit your reply, just like customers can edit their reviews.
- If you come across any inappropriate reviews, like spam, abusive, or offensive content, for any extension, flag it for our review.

Request to void a review

As a publisher, you can appeal to void a review if the issue reported is because of the Marketplace or underlying platform. If the issue is valid, Marketplace admins void the rating. You can **Appeal** from ratings and review section on your extension hub page.

Unpublish an extension

You can unpublish free extensions, if you no longer want to offer them in the Marketplace.

The following scenarios cover when you might want to remove your extension from the Marketplace:

- You developed another extension and no longer want to offer the current one.
- Your extension has a problem, so you want to remove your extension from the Marketplace until you resolve the problem.
- You published your extension as public by mistake.

Certain criteria must be met for an extension to be unpublished or removed:

Expand table

Action	Requirements
Unpublish	Only **free extensions** might be unpublished.
Remove	Your extension must have **zero (0)** installs to be removed.

> **Important**
>
> If you must remove your extension because of legal or security problems, contact Customer Support at the Developer Community. We review the request and manually delete the extension.

1. Select the extension on your publisher page and choose **Unpublish** on the menu.

Your extension is unpublished immediately from the Marketplace, and new users can't install it. Ratings and reviews for your extension stay intact.

To offer your extension again in the Marketplace, choose **Publish** on the menu.

You can also choose to remove your extension completely from the Marketplace if your extension has zero (0) installs. To do so, choose **Remove** on the menu. This action can't be undone.

Related articles

- Develop a web extension
- Extensibility points

Feedback

Was this page helpful? Yes No

Provide product feedback

Extension manifest reference

Article • 09/09/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Every extension has a JSON manifest file that defines basic information about the extension. The file also defines how it can extend and enhance the experience. This article shows you how to create a manifest for your extension to Azure DevOps.

> **Tip**
>
> Check out our newest documentation on extension development using the Azure DevOps Extension SDK.

Create a file named `vss-extension.json` at the root of your extension folder. This file contains required attributes, like the extension's ID and its installation targets, where it can run. It also defines the contributions being made by your extension.

See the following example of a typical manifest:

JSON

```
{
    "manifestVersion": 1,
    "id": "tools",
    "version": "0.1.0",
    "name": "Fabrikam Tools",
    "publisher": "fabrikam",
    "description": "Awesome tools to help you and your team do great things everyday.",
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "icons": {
        "default": "images/fabrikam-logo.png"
    },
    "scopes": [
        "vso.work",
        "vso.code_write",
        "vso.build_execute"
    ],
    "categories": [
        "Azure Boards"
    ],
    "branding": {
        "color": "rgb(34, 34, 34)",
        "theme": "dark"
    },
    "content": {
        "details": {
            "path": "readme.md"
        },
        "license": {
            "path": "eula.md"
        }
    },
    "links": {
        "getstarted": {
            "uri": "https://www.fabrikam-fiber-inc.com/help/getstarted"
        },
```

```
        "support": {
            "uri": "https://www.fabrikam-fiber-inc.com/support"
        }
    },
    "repository": {
        "type": "git",
        "uri": "https://github.com/fabrikam-fiber-inc/myextension"
    },
    "contributions": [
        {
            "id": "showCommits",
            "type": "ms.vss-web.action",
            "description": "Adds a menu action from builds grid to show associated items.",
            "targets": [
                "ms.vss-build-web.completed-build-menu"
            ],
            "properties": {
                "title": "View associated items",
                "uri": "launch.html"
            }
        }
    ],
    "files": [
        {
            "path": "launch.html",
            "addressable": true
        },
        {
            "path": "node_modules/vss-web-extension-sdk/lib",
            "addressable": true,
            "packagePath": "lib"
        }
    ]
}
```

For information about inputs, see ...

Required attributes

These properties are required:

Expand table

Property	Description	Notes
manifestVersion	*A number corresponding to the version of the manifest format.*	Should be `1`.
ID	*The extension's identifier.*	Th ID is a string that must be unique among extensions from the same publisher. It must start with an alphabetic or numeric character and contain 'A' through 'Z', 'a' through 'z', '0' through '9', and '-' (hyphen). Example: `sample-extension`.
version	*A string specifying the version of an extension.*	Should be in the format `major.minor.patch`, for example `0.1.2` or `1.0.0`. You can also add a fourth number for the following format: `0.1.2.3`
name	*A short, human-readable name of the extension. Limited to 200 characters.*	Example: `"Fabrikam Agile Board Extension"`.
publisher	*The identifier of the*	This identifier must match the identifier the extension is published under. See

Property	Description	Notes
	publisher.	Create and manage a publisher.
categories	*Array of strings representing the categories your extension belongs to. At least one category must be provided and there's no limit to how many categories you may include.*	Valid values: `Azure Repos`, `Azure Boards`, `Azure Pipelines`, `Azure Test Plans`, and `Azure Artifacts`. Notes: - Use version >=0.6.3 of the tfx-cli if you're publishing the extension programmatically. - If you're using Azure DevOps Extension Tasks extension to publish, ensure that its version is >= 1.2.8. You might have to approve the extension update because of recent scope changes. - The categories previously mentioned are natively present in Visual Studio Marketplace and Azure DevOps Server 2019 & above. For extensions targeting earlier versions of TFS: - If TFS customers acquire your extension via Visual Studio Marketplace (not local gallery) in connected context, then use the categories that are previously stated. - If you are going to share the extension directly (that is, not via the Visual Studio Marketplace) with a customer using TFS <=2018 then use the following categories instead: Code, Plan and track, Build and release, Test, Collaborate, and Integrate. If you need to share both via Visual Studio Marketplace and directly with a TFS <= 2018 customer then you would need to have 2 extension packages.
targets	*The products and services supported by your integration or extension.* For more information, see installation targets.	An array of objects, where each object has an `id` field indicating one of the following: - `Microsoft.VisualStudio.Services` (extensions that works with Azure DevOps or TFS), - `Microsoft.TeamFoundation.Server` (extension that works with TFS), - `Microsoft.VisualStudio.Services.Integration` (integrations that works with Azure DevOps or TFS), - `Microsoft.TeamFoundation.Server.Integration` (integrations that work with TFS)

Examples of required attributes

JSON

```
{
    "manifestVersion": 1,
    "id": "tools",
    "version": "0.1.0",
    "name": "Fabrikam Tools",
    "publisher": "fabrikam",
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ]
}
```

Optional attributes

Runtime attributes

Expand table

Property	Description	Notes
scopes	*An array of authorization scopes (strings) listing permissions required by your extension.*	For example, `vso.work` and `vs.code_write` indicates your extension needs read-only access to work items and read/write access to source code (and related resource). Scopes are presented to the user when installing your extension. For more information, see the full list of scopes.
demands	*An array of demands (strings) listing the capabilities required by your extension.*	For example, `api-version/3.0` indicates that your extension uses version 3.0 APIs, and so can't run in older products that don't support this version. For more information, see the full list of demands.
baseUri	*(Optional) base URL for all relative URLs specified by the extension's contributions.*	For example: `https://myapp.com/{{account.name}}/`. This property should be left empty if your extension's contents are packaged with your extension.
contributions	*An array of contributions to the system.*	
contributionTypes	*An array of contribution types defined by the extension*	

JSON

```
{
    "scopes": [
        "vso.work",
        "vso.code_write",
        "vso.build_execute"
    ],
    "demands": [
        "api-version/3.0"
    ],
    "contributions": [
        {
            "id": "showCommits",
            "type": "ms.vss-web.action",
            "description": "Adds a menu action from builds grid to show associated items.",
            "targets": [
                "ms.vss-build-web.completed-build-menu"
            ],
            "properties": {
                "title": "View associated items",
                "uri": "launch.html"
            }
        }
    ]
}
```

Discovery attributes

These optional properties help users discover and learn about your extension:

Expand table

Property	Description	Notes
description	*A few sentences describing the extensions. Limited to 200 characters.*	The description should be your extension's "elevator pitch" - a couple of lines to describe your extension in the Marketplace and make people want to install it. See the example below
icons	*Dictionary of icons representing the extension.*	Valid keys: `default` (128x128 pixels) of type BMP, GIF, EXIF, JPG, PNG and TIFF). Other keys such as `large` (512x512 pixels) may be supported in the future. The value of each key is the path to the icon file in the extension
tags	*Array of string tags to help users find your extension.*	Examples: `agile`, `project management`, `task timer`, and so on.
screenshots	*Array of images that couldn't be included in your **content**.*	Screenshots are more valuable when featured in your **content**, and should be used there to help make a quality market details page for your extension. Use **screenshots** for less important images not featured in your **content**. Each image should be 1366x768 pixels. The `path` of each item is the path to the file in the extension.
content	*Dictionary of content files that describe your extension to users.*	*Every* extension should include solid content. This is how you'll show users what your extension can do. Make it rich, consumable, and include screenshots where necessary. Include an `overview.md` file as your base content piece. Each file is assumed to be in GitHub Flavored Markdown format. The `path` of each item is the path to the Markdown file in the extension. Valid keys: `details`. Other keys may be supported in the future.
links	*Dictionary of links that help users learn more about your extension, get support, and move.*	Valid keys: `getstarted` - first steps, how to setup or use. `learn` - deeper content to help users better understand your extension or service. `license` - end-user license agreement. `privacypolicy` - privacy policy for an extension. `support` - get help and support for an extension. The value of each key is an object with a `uri` field, which is the absolute URL of the link
repository	*Dictionary of properties describing the source code repository for the extension*	Valid Keys: `type` - Type of repository. Example: *git*. `uri` - Absolute URL of the repository.
badges	*Array of links to external metadata badges like TravisCI, Appveyor, and so on, from the approved badges sites*	Valid keys: `href` - Link the user navigates to when selecting the badge. `uri` - The absolute URL of the badge image to be displayed. `description` - Description of the badge, to be displayed on hover.
branding	*Dictionary of brand-related properties.*	Valid keys: `color` - primary color of the extension or publisher; can be a hex (#ff00ff), RGB (rgb(100,200,50)), or supported HTML color names (blue). `theme` - complements the color; use *dark* for dark branding colors, or *light* for lighter branding colors.

Mark an extension public

By default, all extensions in the Azure DevOps Marketplace are private. They're only visible to the publisher and accounts shared to by the publisher. If your publisher is verified, you can make your extension public by setting the `Public` flag in your extension manifest:

JSON

```
{
    "galleryFlags": [
        "Public"
    ]
}
```

Or:

```JSON
{
    "public": true
}
```

For more information, see Package/Publish/Install.

Mark an extension to be in preview

If your extension's ready for users on the Marketplace to try, but you're still working out a few bugs or adding function, you can mark it as `preview`:

```JSON
{
    "galleryFlags": [
        "Preview"
    ]
}
```

Mark an extension as paid preview

If you intend to sell your extension on the Marketplace, mark it as *paid preview*. An extension marked *free* can't be changed to *paid*.

```JSON
{
    "galleryFlags": [
        "Paid",
        "Preview"
    ]
}
```

Mark an extension as paid

If you want to sell your extension on the Marketplace, you can mark it with the `Paid` flag and `__BYOLENFORCED` tag (starts with two underscores):

```JSON
{
    "galleryFlags": [
        "Paid"
    ],
     "tags": [
        "__BYOLENFORCED"
    ]
}
```

Both the `Paid` flag and `__BYOLENFORCED` tag need to be present to mark an extension as paid in the Marketplace. Bring-Your-Own-License (BYOL) means the publisher of the extension provides the billing and licensing

mechanism for the extension, as it isn't provided by Microsoft for Azure DevOps extensions. All paid extensions are required to define privacy policy, support policy, and an end-user license agreement. Publishers must provide content for the pricing tab in Marketplace as follows:

JSON

```
{
    "content": {
        "details": {
            "path": "overview.md"
        },
        "pricing": {
            "path": "pricing.md"
        }
    }
}
```

You also need to add a new section in your extension manifest to override paid licensing. In the future, we remove the paid licensing check and no longer require the override. For now, ensure your extension displays as expected. Each override consists of an "ID" and a "behavior." Make the "ID" match the ID of the contributions defined in the manifest.

JSON

```
"licensing": {

      "overrides": [

        { "id": "my-hub", "behavior": " AlwaysInclude" }
      ]
    }
```

If your paid BYOL extension offers a trial period (we recommend so), then you can specify the length of the trial in days:

JSON

```
{
    "galleryproperties": {
        "trialDays": "30"
    }
}
```

Note

If you want to target Azure DevOps, but don't wish to surface a **Download** option for your extension, then add the `__DoNotDownload` tag (starts with two underscores) to the extension manifest. If you're moving an extension from the previously-offered billing & licensing from Microsoft to the BYOL model, then contact us for suitable steps.

Example of more properties

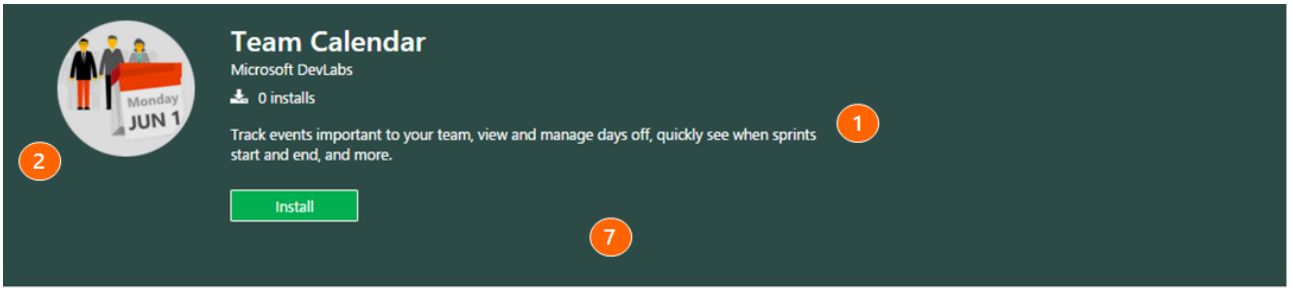
JSON

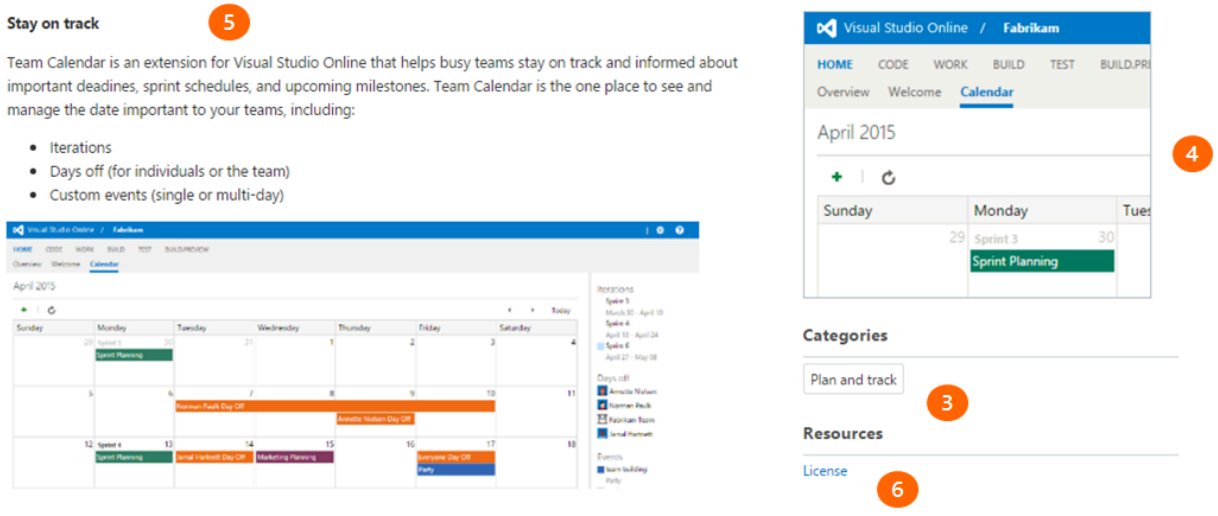
```
{
    "description": "Awesome tools to help you and your team do great things everyday.",
```

```
    "icons": {
        "default": "images/fabrikam-logo.png"
    },
    "categories": [
        "Plan and track"
    ],
    "tags": [
        "working",
        "people person",
        "search"
    ],
    "content": {
        "details": {
            "path": "overview.md"
        },
        "license": {
            "path": "license-terms.md"
        }
    },
    "links": {
        "home": {
            "uri": "https://www.fabrikam-fiber-inc.com"
        },
        "getstarted": {
            "uri": "https://www.fabrikam-fiber-inc.com/help/getstarted"
        },
        "learn": {
            "uri": "https://www.fabrikam-fiber-inc.com/features"
        },
        "support": {
            "uri": "https://www.fabrikam-fiber-inc.com/support"
        },
        "repository": {
            "uri": "https://github.com/fabrikam-fiber-inc/tools"
        },
        "issues": {
            "uri": "https://github.com/fabrikam-fiber-inc/tools/issues"
        }
    },
    "repository": {
        "type": "git",
        "uri": "https://github.com/fabrikam-fiber-inc/tools"
    },
    "badges": [
        {
            "href": "https://travis.ci/fabrikam-fiber-inc/myextension",
            "uri": "https://travis.ci/fabrikam-fiber-inc/myextension.svg?branch=master",
            "description": "TravisCI build for the project"
        },
        {
            "href": "https://ci.appveyor.com/projects/fabrikam-fiber-inc/myextension",
            "uri": "https://ci.appveyor.com/api/projects/status/v1g2sgs2y7tsdxpj4c?svg=true",
            "description": "AppVeyor build for the project"
        }
    ],
    "branding": {
        "color": "rgb(34, 34, 34)",
        "theme": "dark"
    },
    "screenshots": [
        {
            "path": "screenshots/screen1.png"
        },
        {
            "path": "screenshots/screen2.png"
        }
    ]
}
```

Details page example

- 1 - description
- 2 - icon
- 3 - categories
- 4 - screenshots
- 5 - content (details)
- 6 - links
- 7 - branding





Marketplace Q & A - CustomerQnASupport property

All extensions on the Visual Studio Marketplace have a Questions and Answers (Q & A) section to allow one-on-one public conversations between extension users and publishers. Publishers can choose between Marketplace Questions and Answers (Q&A), GitHub issues, or a custom Q&A URL. You can disable Q&A in the Marketplace using the CustomerQnASupport property in the manifest.

Default experience (No changes to manifest are required)

- For extensions with a GitHub repository, Marketplace redirects users in the Q&A section to the associated GitHub issues.
- For extensions without a GitHub repository, Marketplace Q&A is enabled.

For a different experience than one of the default options, use the **CustomerQnASupport** property in the manifest.

JSON

```
{
    "CustomerQnASupport": {
```

```
        "enablemarketplaceqna": true,
        "url": "http://uservoice.visualstudio.com"
    }
}
```

Properties

Properties for the Customer Q & A Support section:

- **enablemarketplaceqna** - boolean field, set to `true` for marketplace, or custom Q&A; false for disabling Q&A
- **url** - string, URL for custom Q&A

Examples showing usage of Q & A support

Example: Extension using custom Q & A

JSON

```
{
     "CustomerQnASupport": {
        "enablemarketplaceqna":"true",
        "url": "http://uservoice.visualstudio.com"
    }
}
```

Example: Extension with GitHub repository but using Marketplace Q & A instead of GitHub issues

JSON

```
{
     "CustomerQnASupport": {
        "enablemarketplaceqna":"true"
    }
}
```

Example: Extension disabling Q & A section

JSON

```
{
     "CustomerQnASupport": {
        "enablemarketplaceqna":"false"
    }
}
```

Scopes

In your extension, you can define one or more scopes. These scopes determine which resources your extension can access and the operations permitted to perform on those resources. The scopes you specify in your

extension manifest are the scopes set on access tokens issued to your extension. For more information, see Auth and security.

If no scopes are specified, extensions are only provided access to user profile and extension data.

Supported scopes

Expand table

Category	Scope	Name	Description	Inherits From
Advanced Security	`vso.advsec`	AdvancedSecurity (read)	Grants the ability to read alerts, result instances, analysis result instances.	
	`vso.advsec_write`	AdvancedSecurity (read and write)	Grants the ability to upload analyses in sarif	`vso.advsec`
	`vso.advsec_manage`	AdvancedSecurity (read, write, and manage)	Grants the ability to upload analyses in sarif	`vso.advsec_write`
Agent Pools	`vso.agentpools`	Agent Pools (read)	Grants the ability to view tasks, pools, queues, agents, and currently running or recently completed jobs for agents.	
	`vso.agentpools_manage`	Agent Pools (read, manage)	Grants the ability to manage pools, queues, and agents.	`vso.agentpools`
	`vso.environment_manage`	Environment (read, manage)	Grants the ability to manage pools, queues, agents, and environments.	`vso.agentpools_manage`
Analytics	`vso.analytics`	Analytics (read)	Grants the ability to query analytics data.	

Category	Scope	Name	Description	Inherits From
Auditing	`vso.auditlog`	Audit Log (read)	Grants the ability to read the auditing log to users.	
	`vso.auditstreams_manage`	Audit Streams (read)	Grants the ability to manage auditing streams to users.	`vso.auditlog`
Build	`vso.build`	Build (read)	Grants the ability to access build artifacts, including build results, definitions, and requests, and the ability to receive notifications about build events via service hooks.	`vso.hooks_write`
	`vso.build_execute`	Build (read and execute)	Grants the ability to access build artifacts, including build results, definitions, and requests, and the ability to queue a build, update build properties, and the ability to receive notifications about build events via service hooks.	`vso.build`
Code	`vso.code`	Code (read)	Grants the ability to read source code and metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to	`vso.hooks_write`

Category	Scope	Name	Description	Inherits From
			search code and get notified about version control events via service hooks.	
	`vso.code_write`	Code (read and write)	Grants the ability to read, update, and delete source code, access metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to create and manage pull requests and code reviews and to receive notifications about version control events via service hooks.	`vso.code`
	`vso.code_manage`	Code (read, write, and manage)	Grants the ability to read, update, and delete source code, access metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to create and manage code repositories, create and manage pull requests and code reviews, and to receive notifications about version control events	`vso.code_write`

Category	Scope	Name	Description	Inherits From
			via service hooks.	
	`vso.code_full`	Code (full)	Grants full access to source code, metadata about commits, changesets, branches, and other version control artifacts. Also grants the ability to create and manage code repositories, create and manage pull requests and code reviews, and to receive notifications about version control events via service hooks. Also includes limited support for Client OM APIs.	`vso.code_manage`
	`vso.code_status`	Code (status)	Grants the ability to read and write commit and pull request status.	
Connected Server	`vso.connected_server`	Connected Server	Grants the ability to access endpoints needed from an on-premises connected server.	
Entitlements	`vso.entitlements`	Entitlements (Read)	Provides read only access to licensing entitlements endpoint to get account entitlements.	

Category	Scope	Name	Description	Inherits From
	`vso.memberentitlementmanagement`	MemberEntitlement Management (read)	Grants the ability to read users, their licenses as well as projects and extensions they can access.	
	`vso.memberentitlementmanagement_write`	MemberEntitlement Management (write)	Grants the ability to manage users, their licenses as well as projects and extensions they can access.	`vso.memberentitlementmanagement`
Extensions	`vso.extension`	Extensions (read)	Grants the ability to read installed extensions.	`vso.profile`
	`vso.extension_manage`	Extensions (read and manage)	Grants the ability to install, uninstall, and perform other administrative actions on installed extensions.	`vso.extension`
	`vso.extension.data`	Extension data (read)	Grants the ability to read data (settings and documents) stored by installed extensions.	`vso.profile`
	`vso.extension.data_write`	Extension data (read and write)	Grants the ability to read and write data (settings and documents) stored by installed extensions.	`vso.extension.data`
Github Connections	`vso.githubconnections`	GitHub Connections (read)	Grants the ability to read GitHub connections and GitHub repositories data.	

Category	Scope	Name	Description	Inherits From
	`vso.githubconnections_manage`	GitHub Connections (read and manage)	Grants the ability to read and manage GitHub connections and GitHub repositories data	`vso.githubconnections`
Graph & identity	`vso.graph`	Graph (read)	Grants the ability to read user, group, scope, and group membership information.	
	`vso.graph_manage`	Graph (manage)	Grants the ability to read user, group, scope and group membership information, and to add users, groups, and manage group memberships.	`vso.graph`
	`vso.identity`	Identity (read)	Grants the ability to read identities and groups.	
	`vso.identity_manage`	Identity (manage)	Grants the ability to read, write, and manage identities and groups.	`vso.identity`
Machine Group	`vso.machinegroup_manage`	Deployment group (read, manage)	Provides ability to manage deployment group and agent pools.	`vso.agentpools_manage`
Marketplace	`vso.gallery`	Marketplace	Grants read access to public and private items and publishers.	`vso.profile`
	`vso.gallery_acquire`	Marketplace (acquire)	Grants read access and the ability to acquire items.	`vso.gallery`

Category	Scope	Name	Description	Inherits From
	`vso.gallery_publish`	Marketplace (publish)	Grants read access and the ability to upload, update, and share items.	`vso.gallery`
	`vso.gallery_manage`	Marketplace (manage)	Grants read access and the ability to publish and manage items and publishers.	`vso.gallery_publish`
Notifications	`vso.notification`	Notifications (read)	Provides read access to subscriptions and event metadata, including filterable field values.	`vso.profile`
	`vso.notification_write`	Notifications (write)	Provides read and write access to subscriptions and read access to event metadata, including filterable field values.	`vso.notification`
	`vso.notification_manage`	Notifications (manage)	Provides read, write, and management access to subscriptions and read access to event metadata, including filterable field values.	`vso.notification_write`
	`vso.notification_diagnostics`	Notifications (diagnostics)	Provides access to notification-related diagnostic logs and provides the ability to enable diagnostics for	`vso.notification`

Category	Scope	Name	Description	Inherits From
			individual subscriptions.	
Packaging	`vso.packaging`	Packaging (read)	Grants the ability to read feeds and packages.	`vso.profile`
	`vso.packaging_write`	Packaging (read and write)	Grants the ability to create and read feeds and packages.	`vso.packaging`
	`vso.packaging_manage`	Packaging (read, write, and manage)	Grants the ability to create, read, update, and delete feeds and packages.	`vso.packaging_write`
Pipeline Resources	`vso.pipelineresources_use`	Pipeline Resources (use)	Grants the ability to approve a pipeline's request to use a protected resource: agent pool, environment, queue, repository, secure files, service connection, and variable group.	
	`vso.pipelineresources_manage`	Pipeline Resources (use and manage)	Grants the ability to manage a protected resource or a pipeline's request to use a protected resource: agent pool, environment, queue, repository, secure files, service connection, and variable group.	`vso.pipelineresources_manage`
Project and Team	`vso.project`	Project and team (read)	Grants the ability to read	

Category	Scope	Name	Description	Inherits From
			projects and teams.	
	`vso.project_write`	Project and team (read and write)	Grants the ability to read and update projects and teams.	`vso.project`
	`vso.project_manage`	Project and team (read, write and manage)	Grants the ability to create, read, update, and delete projects and teams.	`vso.project_write`
Release	`vso.release`	Release (read)	Grants the ability to read release artifacts, including releases, release definitions and release environment.	`vso.profile`
	`vso.release_execute`	Release (read, write and execute)	Grants the ability to read and update release artifacts, including releases, release definitions and release environment, and the ability to queue a new release.	`vso.release`
	`vso.release_manage`	Release (read, write, execute and manage)	Grants the ability to read, update, and delete release artifacts, including releases, release definitions and release environment, and the ability to queue and approve a new release.	`vso.release_manage`
Secure Files	`vso.securefiles_read`	Secure Files (read)	Grants the ability to read	

Category	Scope	Name	Description	Inherits From
			secure files.	
	`vso.securefiles_write`	Secure Files (read, create)	Grants the ability to read and create secure files.	`vso.securefiles_read`
	`vso.securefiles_manage`	Secure Files (read, create, and manage)	Grants the ability to read, create, and manage secure files.	`vso.securefiles_write`
Security	`vso.security_manage`	Security (manage)	Grants the ability to read, write, and manage security permissions.	
Service Connections	`vso.serviceendpoint`	Service Endpoints (read)	Grants the ability to read service endpoints.	`vso.profile`
	`vso.serviceendpoint_query`	Service Endpoints (read and query)	Grants the ability to read and query service endpoints.	`vso.serviceendpoint`
	`vso.serviceendpoint_manage`	Service Endpoints (read, query and manage)	Grants the ability to read, query, and manage service endpoints.	`vso.serviceendpoint_query`
Service Hooks	`vso.hooks`	Service hooks (read)	Grants the ability to read service hook subscriptions and metadata, including supported events, consumers, and actions. (No longer public.)	`vso.profile`
	`vso.hooks_write`	Service hooks (read and write)	Grants the ability to create and update service hook subscriptions and read metadata, including supported	`vso.hooks`

Category	Scope	Name	Description	Inherits From
			events, consumers, and actions. (No longer public.)	
	`vso.hooks_interact`	Service hooks (interact)	Grants the ability to interact and perform actions on events received via service hooks. (No longer public.)	`vso.profile`
Settings	`vso.settings`	Settings (read)	Grants the ability to read settings.	
	`vso.settings_write`	Settings (read and write)	Grants the ability to create and read settings.	
Symbols	`vso.symbols`	Symbols (read)	Grants the ability to read symbols.	`vso.profile`
	`vso.symbols_write`	Symbols (read and write)	Grants the ability to read and write symbols.	`vso.symbols`
	`vso.symbols_manage`	Symbols (read, write and manage)	Grants the ability to read, write, and manage symbols.	`vso.symbols_write`
Task Groups	`vso.taskgroups_read`	Task Groups (read)	Grants the ability to read task groups.	
	`vso.taskgroups_write`	Task Groups (read, create)	Grants the ability to read and create task groups.	`vso.taskgroups_read`
	`vso.taskgroups_manage`	Task Groups (read, create and manage)	Grants the ability to read, create and manage taskgroups.	`vso.taskgroups_write`
Team Dashboard	`vso.dashboards`	Team dashboards (read)	Grants the ability to read team dashboard information.	

Category	Scope	Name	Description	Inherits From
	`vso.dashboards_manage`	Team dashboards (manage)	Grants the ability to manage team dashboard information.	`vso.dashboards`
Test Management	`vso.test`	Test management (read)	Grants the ability to read test plans, cases, results and other test management related artifacts.	`vso.profile`
	`vso.test_write`	Test management (read and write)	Grants the ability to read, create, and update test plans, cases, results and other test management related artifacts.	`vso.test`
Threads	`vso.threads_full`	PR threads	Grants the ability to read and write to pull request comment threads.	
Tokens	`vso.tokens`	Delegated Authorization Tokens	Grants the ability to manage delegated authorization tokens to users.	
	`vso.tokenadministration`	Token Administration	Grants the ability to manage (view and revoke) existing tokens to organization administrators.	
User Profile	`vso.profile`	User profile (read)	Grants the ability to read your profile, accounts, collections, projects, teams, and other top-level	

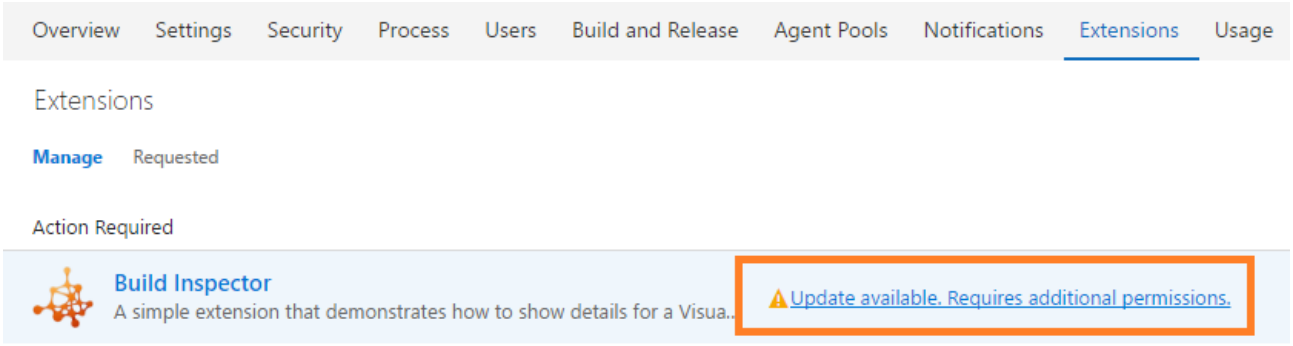
Category	Scope	Name	Description	Inherits From
			organizational artifacts.	
	`vso.profile_write`	User profile (write)	Grants the ability to write to your profile.	`vso.profile`
Variable Groups	`vso.variablegroups_read`	Variable Groups (read)	Grants the ability to read variable groups.	
	`vso.variablegroups_write`	Variable Groups (read, create)	Grants the ability to read and create variable groups.	`vso.variablegroups_read`
	`vso.variablegroups_manage`	Variable Groups (read, create and manage)	Grants the ability to read, create and manage variable groups.	`vso.variablegroups_write`
Wiki	`vso.wiki`	Wiki (read)	Grants the ability to read wikis, wiki pages and wiki attachments. Also grants the ability to search wiki pages.	
	`vso.wiki_write`	Wiki (read and write)	Grants the ability to read, create and updates wikis, wiki pages and wiki attachments.	`vso.wiki`
Work Items	`vso.work`	Work items (read)	Grants the ability to read work items, queries, boards, area and iterations paths, and other work item tracking related metadata. Also grants the ability to execute queries, search work items and to receive	`vso.hooks_write`

Category	Scope	Name	Description	Inherits From
			notifications about work item events via service hooks.	
	`vso.work_write`	Work items (read and write)	Grants the ability to read, create, and update work items and queries, update board metadata, read area and iterations paths other work item tracking related metadata, execute queries, and to receive notifications about work item events via service hooks.	`vso.work`
	`vso.work_full`	Work items (full)	Grants full access to work items, queries, backlogs, plans, and work item tracking metadata. Also provides the ability to receive notifications about work item events via service hooks.	`vso.work_write`
User Impersonation	`user_impersonation`	User Impersonation	Have full access to Visual Studio Team Services REST APIs. **Request and/or consent this scope with caution as it is very powerful!**	

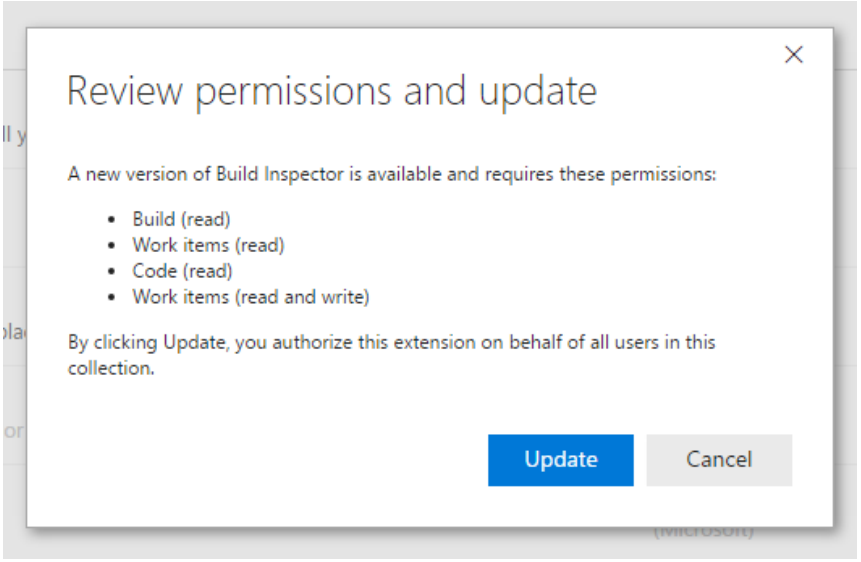
Changing scope of published extension

You can change the scope of a published extension. If you previously installed your extension (and authorized the previous set of scopes), authorize the new scopes before you can upgrade to the newest version.

The **Action Required** section of the Extension settings hub shows a user that, if any, installed extensions require authorization:



An administrator can then review and authorize the new set of scopes:



Installation targets

As the name implies, installation targets define the products and services where you can install your extension. `Microsoft.VisualStudio.Services` is the most common installation target and indicates that the extension can be installed into Azure DevOps.

The installation targets for an extension or integration are specified via the `targets` field in the manifest.

Supported identifiers for **extensions**:

- `Microsoft.VisualStudio.Services.Cloud`: installs into Azure DevOps Services
- `Microsoft.TeamFoundation.Server`: installs into Azure DevOps Server
- `Microsoft.VisualStudio.Services`: installs into both. Shortcut for `Microsoft.VisualStudio.Services.Cloud` and `Microsoft.TeamFoundation.Server` version `[14.2,)`

Supported identifiers for **integrations**:

- `Microsoft.VisualStudio.Services.Cloud.Integration`: integrates with Azure DevOps Services

- `Microsoft.TeamFoundation.Server.Integration`: integrates with Azure DevOps Server
- `Microsoft.VisualStudio.Services.Integration`: integrates with both. Shortcut for `Microsoft.VisualStudio.Services.Cloud.Integration` and `Microsoft.TeamFoundation.Server.Integration`

For more information, see Extensibility points.

Examples of installation targets

Example: Extension that works with Azure DevOps

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ]
}
```

Example: Extension that works only with Azure DevOps Services

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services.Cloud"
        }
    ]
}
```

Installation targets can also be used in the manifest of integrations. For example, products, apps, or tools that work with, but don't install into Azure DevOps.

Example: Integration that works with Azure DevOps

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services.Integration"
        }
    ]
}
```

Example: Integration that only works with Azure DevOps Server

JSON

```
{
    "targets": [
        {
```

```
            "id": "Microsoft.TeamFoundation.Server.Integration"
        }
    ]
}
```

Installation target versions

Some installation target identifiers, like `Microsoft.TeamFoundation.Server` and `Microsoft.TeamFoundation.Server.Integration`, support an optional version range. This optional version range further clarifies the supported releases the extension or integration is supported on.

The version or version range is specified via the `version` field on the installation target object. This value can be either:

- A specific version, for example: `15.0` (2017 RTM only)
- A range of supported versions, for example: `[14.0)` (2015 RTM and later), `[14.3,15.1]` (2015 Update 3 through 2017 Update 1). Range values are refined using:
  - `[`: minimum version inclusive
  - `]`: maximum version inclusive
  - `(`: minimum version exclusive
  - `)`: maximum version exclusive

Version numbers for Azure DevOps Server:

Expand table

Release	Releases	Version
2010	All releases	10.0
2012	All releases	11.0
2013	RTM and updates	12.0, 12.1, 12.2, 12.3, 12.4
2015	RTM and updates	14.0, 14.1, 14.2, 14.3
2017	RTM and updates	15.0, 15.1
2018	RTM and updates	16.0
2019	RTM and updates	17.0
2020	RTM and updates	18.0

Examples showing versions

Example: Extension that works with Azure DevOps

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services.Cloud"
        },
        {
```

```
            "id": "Microsoft.TeamFoundation.Server",
            "version": "[15.0,)"
        }
    ]
}
```

Shortcuts

`Microsoft.VisualStudio.Services` is a shortcut for Azure DevOps.

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ]
}
```

is equivalent to:

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services.Cloud"
        },
        {
            "id": "Microsoft.TeamFoundation.Server",
            "version": "[14.2,)"
        }
    ]
}
```

Using installation targets and demands

Installation targets and demands are used together to present users with a correct view of the products/services your extension or integration is compatible with. For example, specifying an installation target of `Microsoft.VisualStudio.Services` with a demand of `api-version/3.0` means the extension works with Azure DevOps.

> **Tip**
>
> For more information on REST APIs, see the REST API Reference.

Example: Extension that uses version 3.0 APIs

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
```

```
    ],
    "demands": [
        "api-version/3.0"
    ]
}
```

Resolves to the following installation targets:

1. `Microsoft.VisualStudio.Services.Cloud`
2. `Microsoft.TeamFoundation.Server`, version: `[15.0,)`

Example: Integration that uses version 2.0 APIs

JSON

```
{
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services.Integration"
        }
    ],
    "demands": [
        "api-version/2.0"
    ]
}
```

Resolves to the following installation targets:

1. `Microsoft.VisualStudio.Services.Cloud.Integration`
2. `Microsoft.TeamFoundation.Server.Integration`, version: `[14.0,)`

Demands

Demands let you specify capabilities and other features required by your extension. You can use these demands to limit where your extension can be published or installed.

Demands get used by the Visual Studio Marketplace to list the products and environments your extension is compatible with, which helps customers understand whether your extension works with their version of Azure DevOps, for example.

See the following example of how demands get specified in the extension manifest.

JSON

```
{
    "demands": [
        "api-version/3.0",
        "contribution/ms.vss-dashboards-web.widget-catalog"
    ]
}
```

In this example, the extension demands version 3.0 of the APIs, which means it can only be installed to Azure DevOps. It also requires the `ms.vss-dashboards-web` extension (and its `widget-catalog` contribution) to be installed (and enabled) in the collection before your extension can be installed.

Supported demands

Expand table

Type	Description	Checked at publish?	Checked at install?
`environment/cloud`	Requires running in a cloud environment	Yes	Yes
`environment/onprem`	Requires running in an on-premises environment	Yes	Yes
`api-version/{version}`	Requires a specific API version (minimum)	No	Yes
`extension/{id}`	Requires a specific extension be installed/enabled	No	Yes
`contribution/{id}`	Requires a specific contribution be available	No	Yes
`contributionType/{id}`	Requires a specific contribution type be available	No	Yes

> **Note**
>
> - Use `environment/cloud` and `environment/onprem` only when your extension has topology-related requirements that require running in that particular environment.
> - `extension`, `contribution`, and `contributionType` demands are evaluated at install time, and requires that the specified extension is already installed and enabled in the organization/collection.

Files

The `files` section is where you reference any files you wish to include in your extension. You can add both folders and individual files:

JSON

```
{
    "files": [
        {
            "path": "hello-world.html", "addressable": true
        },
        {
            "path": "scripts", "addressable": true
        },
        {
            "path": "images/logo.png", "addressable": true, "packagePath": "/"
        }
    ]
}
```

Properties

Properties for the Files section:

- **path** - Path to resource on disk, which can be relative to your root directory.
- **addressable** – (optional) Set to **true** if you want your file to be URL-addressable. Defaults to **false**.
- **packagePath** – (optional) Path to the resource within the package. Defaults to the relative path on disk from your root directory.

- **contentType** – (optional) MIME type of the file. Defaults to a best guess based on the file extension and OS settings.
- **assetType** – (optional) Specify the value of the Type attribute of the asset entry in the VSIX manifest. Can also be an array of strings, in which case multiple asset entries get added for this file. Defaults to the packagePath.
- **lang** – (optional) Language of this asset. Localized files are served based on the Accept-Language header. Leave blank to signify this file is in the default (or fallback) language. Localized versions of the same file should have the same assetType.

Contributions

Each contribution entry has the following properties:

- **id** - A reference ID (string) for the contribution. Make each contribution ID unique within an extension. See referencing contributions and types.
- **type** - The ID of the contributionType of this contribution.
- **description** - (Optional) A string describing what the contribution is providing.
- **targets** - An array of contribution IDs that the contribution is targeting (contributing to). See Targeting contributions.
- **properties** - (Optional) An object that includes properties for the contribution as defined in the contribution type.

For more information, see the contribution model overview.

Contribution types

Each contribution entry has the following properties:

- **id** - A reference ID (string) for the contribution type. Make each contribution type ID unique within an extension. See referencing contributions and types.
- **name** - The friendly name of the contribution type.
- **description** - (Optional) A string describing in more detail what the contribution type is for.
- **properties** - (Optional) A dictionary that maps property names to property descriptions. These properties describe the required and optional properties that contributions of this type can use.

Property descriptions have the following properties:

- **description** - (Optional) A string describing what the property is used for.
- **required** - (Optional) A boolean value, which if true indicates that the property is required for all contributions of this type.
- **type** - The type of value that the property can have, which could be string, uri, guid, boolean, integer, double, dateTime, array, or object.

For more information, see the contribution model overview.

Referencing contributions and types

Use unique identifiers to reference contributions and contribution types. Reference *types* with the `type` property, and reference other contributions with the `targets` property.

- A *full* contribution reference includes the publisher identifier, extension identifier, and contribution/type identifier, separated by a dot (.). For example, `ms.vss-web.hub` is the full identifier for the contribution with

identifier of "hub" in the "vss-web" extension published by the "ms" (Microsoft) publisher.
- *Relative* contribution references might get used within an extension manifest for a contribution's reference to another contribution or contribution type within that same extension. In this case, the publisher and extension identifiers are NOT included, and the identifier is a dot (.) followed by the contribution identifier. For example, ".hub" might be used within the "vss-web" extension mentioned previously as a shortcut for "ms.vss-web.hub."

Targeting contributions

Some contributions act as containers targeted by other contributions.

- Hub contributions can target Hub Groups. When a page is rendered, the web UI shows all Hub contributions that target the selected hub group. Hub groups target a hub group collection, which defines a set of hub groups that show up in a given navigational area, for example, project-level admin pages.
- Different types of contributions can target menus: action, hyperlink-action, and action-provider. Actions and hyperlink-actions provide single menu item entries. An action-provider can provide multiple dynamic menu items. For a given menu, items are aggregated across all contributions (of any of these types) that target that specific menu contribution.

Adding a hub icon

For information on adding an icon to your hub, check out the hub icon guidance.

Supported badge services

The Marketplace only supports badges from the following trusted services:

- api.travis-ci.org/
- badge.fury.io/
- badges.frapsoft.com/
- badges.gitter.im/
- badges.greenkeeper.io/
- cdn.travis-ci.org/
- ci.appveyor.com/
- codeclimate.com/
- codecov.io/
- coveralls.io/
- david-dm.org/
- gemnasium.com/
- img.shields.io/
- isitmaintained.com/
- marketplace.visualstudio.com/
- snyk.io/
- travis-ci.com/
- travis-ci.org/
- vsmarketplacebadges.dev/
- bithound.io/
- deepscan.io/
- githost.io/
- gitlab.com/

- opencollective.co/

> **Note**
>
> Replace "vsmarketplacebadge.apphb.com" with "vsmarketplacebadges.dev".

To show a badge from another service, contact Customer Support at the Developer Community.

Example manifest

The following extension contributes an action to the completed builds context menu and a hub to the Build hub group:

JSON

```
{
    "manifestVersion": 1,
    "id": "tools",
    "version": "0.1.0",
    "name": "Fabrikam Tools",
    "publisher": "fabrikam",
    "description": "Awesome tools to help you and your team do great things everyday.",
    "targets": [
        {
            "id": "Microsoft.VisualStudio.Services"
        }
    ],
    "demands": [
        "api-version/3.0"
    ],
    "icons": {
        "default": "images/fabrikam-logo.png"
    },
    "scopes": [
        "vso.work",
        "vso.code_write"
    ],
    "categories": [
        "Plan and track"
    ],
    "tags": [
        "working",
        "people person",
        "search"
    ],
    "branding": {
        "color": "rgb(34, 34, 34)",
        "theme": "dark"
    },
    "screenshots": [
        {
            "path": "screenshots/screen1.png"
        },
        {
            "path": "screenshots/screen2.png"
        }
    ],
    "content": {
        "details": {
            "path": "overview.md"
        },
        "license": {
```

```
            "path": "eula.md"
        }
    },
    "links": {
        "home": {
            "uri": "https://www.fabrikam-fiber-inc.com"
        },
        "getstarted": {
            "uri": "https://www.fabrikam-fiber-inc.com/help/getstarted"
        },
        "learn": {
            "uri": "https://www.fabrikam-fiber-inc.com/features"
        },
        "support": {
            "uri": "https://www.fabrikam-fiber-inc.com/support"
        },
        "repository": {
            "uri": "https://github.com/fabrikam-fiber-inc/tools"
        },
        "issues": {
            "uri": "https://github.com/fabrikam-fiber-inc/tools/issues"
        }
    },
    "repository": {
        "type": "git",
        "uri": "https://github.com/fabrikam-fiber-inc/myextension"
    },
    "badges": [
        {
            "href": "https://travis.ci/fabrikam-fiber-inc/myextension",
            "uri": "https://travis.ci/fabrikam-fiber-inc/myextension.svg?branch=master",
            "description": "TravisCI build for the project"
        },
        {
            "href": "https://ci.appveyor.com/projects/fabrikam-fiber-inc/myextension",
            "uri": "https://ci.appveyor.com/api/projects/status/vlg2sgs2y7tsdxpj4c?svg=true",
            "description": "AppVeyor build for the project"
        }
    ],
    "contributions": [
        {
            "id": "showCommits",
            "type": "ms.vss-web.action",
            "description": "Adds a menu action from builds grid to show associated items.",
            "targets": [
                "ms.vss-build-web.completed-build-menu"
            ],
            "properties": {
                "title": "View associated items",
                "uri": "launch.html"
            }
        }
    ]
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback

Reference - integrate custom build tasks into extensions

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Use this reference when you want to create and integrate custom build pipeline tasks with extensions in Azure DevOps.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Custom tasks

Tasks are the building blocks for defining automation in a build or release pipeline. We offer many built-in tasks to enable fundamental build and deployment scenarios. We recommend that you review these scenarios before building your own. If the existing tasks don't satisfy your needs, you can build a custom task.

Custom build task JSON

When you're creating a custom build or release task with an extension, your extension includes a `task.json` file for each build or release task. The `task.json` file describes the build or release task. The build system uses it to render configuration options to the user and to know which scripts to execute at build time.

The build/release task SDK documentation is on GitHub. Specifically, you may be interested in the task.json schema.

Bundle multiple versions of build/release tasks within one extension

You can include multiple versions of a build or release task within your extension. Roll out future versions of your extension without interrupting service of users on older

versions. The following information shows the layout for having multiple versions in one extension.

Traditional extension layout

- extensionManifest.json
- extensionIcon.png
- Task1
  - task.json
  - icon.png
  - taskScript.ps1

> **Note**
>
> When you're setting up a task icon, ensure the following is true.
>
> - The icon name is icon.png
> - The icon size is 32x32 pixels
> - The icon is in the same location as the `task.json` file

Multiple versions layout

> **Note**
>
> The code looks for the `task.json` file inside of the task folder and then one level deeper. If one isn't found in either level, you see an error message.

- extensionManifest.json
- extensionIcon.png
- Task1
  - Task1V1
    - task.json
    - icon.png
    - taskScript.ps1
  - Task1V2
    - task.json
    - icon.png
    - taskScript.ps1
- Task2

- Task2V1
  - task.json
  - icon.png
  - taskScript.ps1
- Task2V2
  - task.json
  - icon.png
  - taskScript.ps1

> **Tip**
>
> To ensure the `_build/Tasks/ssrsfilesdeploy` folder contains the V1 and V2 contents, set `matchCopy(item, srcPath, destPath, { noRecurse:` to `false` in the `make-util.js` file.

Related articles

- Add a custom pipelines task extension
- Server Task GitHub Documentation
- Build/Release Task Examples

Pipeline decorator expression context

Article • 11/08/2024

Azure DevOps Services

Pipeline decorators have access to context about the pipeline in which they run. As a pipeline decorator author, you can use this context to make decisions about the decorator's behavior. The information available in context is different for pipelines and for release. Also, decorators run after task names are resolved to task globally unique identifiers (GUIDs). When your decorator wants to reference a task, it should use the GUID rather than the name or keyword.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

> **Note**
>
> Pipeline decorators are used when building **web extensions**. These examples are not designed to work in YAML pipelines.

Resources

Pipeline resources are available on the `resources` object.

Repositories

Currently, there's only one key: `repositories`. `repositories` is a map from repo ID to information about the repository.

In a designer build, the primary repo alias is `__designer_repo`. In a YAML pipeline, the primary repo is called `self`. In a release pipeline, repositories aren't available. Release artifact variables are available.

For example, to print the name of the `self` repo in a YAML pipeline:

```
```

```
steps:
- script: echo ${{ resources.repositories['self'].name }}
```

Repositories contain these properties:

JavaScript

```
resources['repositories']['self'] =
{
    "alias": "self",
    "id": "<repo guid>",
    "type": "Git",
    "version": "<commit hash>",
    "name": "<repo name>",
    "project": "<project guid>",
    "defaultBranch": "<default ref of repo, like 'refs/heads/main'>",
    "ref": "<current pipeline ref, like 'refs/heads/topic'>",
    "versionInfo": {
        "author": "<author of tip commit>",
        "message": "<commit message of tip commit>"
    },
    "checkoutOptions": {}
}
```

Job

Job details are available on the `job` object.

The data looks similar to:

JavaScript

```
job =
{
    "steps": [
        {
            "environment": null,
            "inputs": {
                "script": "echo hi"
            },
            "type": "Task",
            "task": {
                "id": "d9bafed4-0b18-4f58-968d-86655b4d2ce9",
                "name": "CmdLine",
                "version": "2.146.1"
            },
            "condition": null,
            "continueOnError": false,
            "timeoutInMinutes": 0,
```

```
            "id": "5c09f0b5-9bc3-401f-8cfb-09c716403f48",
            "name": "CmdLine",
            "displayName": "CmdLine",
            "enabled": true
        }
    ]
}
```

For instance, to conditionally add a task only if it doesn't already exist:

YAML

```
- ${{ if not(containsValue(job.steps.*.task.id, 'f3ab91e7-bed6-436a-b651-399a66fe6c2a')) }}:
  - script: echo conditionally inserted
```

Variables

Pipeline variables are also available.

For instance, if the pipeline had a variable called `myVar`, its value would be available to the decorator as `variables['myVar']`.

For example, to give a decorator an opt-out, we could look for a variable. Pipeline authors who wish to opt out of the decorator can set this variable, and the decorator isn't injected. If the variable isn't present, then the decorator is injected as usual.

my-decorator.yml

YAML

```
- ${{ if ne(variables['skipInjecting'], 'true') }}:
  - script: echo Injected the decorator
```

Then, in a pipeline in the organization, the author can request the decorator not to inject itself.

pipeline-with-opt-out.yml

YAML

```
variables:
  skipInjecting: true
```

```yaml
steps:
- script: echo This is the only step. No decorator is added.
```

Task names and GUIDs

Decorators run after tasks already turned into GUIDs. Consider the following YAML:

YAML

```yaml
steps:
- checkout: self
- bash: echo This is the Bash task
- task: PowerShell@2
  inputs:
    targetType: inline
    script: Write-Host This is the PowerShell task
```

Each of those steps maps to a task. Each task has a unique GUID. Task names and keywords map to task GUIDs before decorators run. If a decorator wants to check for the existence of another task, it must search by task GUID rather than by name or keyword.

For normal tasks (which you specify with the `task` keyword), you can look at the task's `task.json` to determine its GUID. For special keywords like `checkout` and `bash` in the previous example, you can use the following GUIDs:

Expand table

Keyword	GUID	Task Name
`checkout`	`6D15AF64-176C-496D-B583-FD2AE21D4DF4`	n/a, see note
`bash`	`6C731C3C-3C68-459A-A5C9-BDE6E6595B5B`	Bash
`script`	`D9BAFED4-0B18-4F58-968D-86655B4D2CE9`	CmdLine
`powershell`	`E213FF0F-5D5C-4791-802D-52EA3E7BE1F1`	PowerShell
`pwsh`	`E213FF0F-5D5C-4791-802D-52EA3E7BE1F1`	PowerShell
`publish`	`ECDC45F6-832D-4AD9-B52B-EE49E94659BE`	PublishPipelineArtifact
`download`	`30f35852-3f7e-4c0c-9a88-e127b4f97211`	DownloadPipelineArtifact

After task names and keywords resolve, the previous YAML becomes:

YAML

```yaml
steps:
- task: 6D15AF64-176C-496D-B583-FD2AE21D4DF4@1
  inputs:
    repository: self
- task: 6C731C3C-3C68-459A-A5C9-BDE6E6595B5B@3
  inputs:
    targetType: inline
    script: echo This is the Bash task
- task: E213FF0F-5D5C-4791-802D-52EA3E7BE1F1@2
  inputs:
    targetType: inline
    script: Write-Host This is the PowerShell task
```

Tip

You can find each of these GUIDs in the `task.json` for the corresponding in-box task. The only exception is `checkout`, which is a native capability of the agent. Its GUID is built into the Azure Pipelines service and agent.

Feedback

Was this page helpful? Yes No

Provide product feedback

Service endpoint authentication schemes

Article • 01/09/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Learn how to set the credentials in the HTTP request header when you're calling the external endpoint. Azure DevOps can then connect to the external service using the credentials. Azure DevOps supports a closed set of authentication schemes utilized by a custom service endpoint type. Azure DevOps interprets the authentication scheme that's used in any custom endpoint & support connection to the external service.

See the following authentication schemes that are part of the closed set.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Basic authentication

As a security measure, we recommend using service principals & managed identities over basic authentication. For more information, see Use service principals & managed identities.

Token-based authentication

This scheme takes one input - API Token (confidential)

Default authentication header used is: {{endpoint.apitoken}}

JSON

```
{
    "id": "endpoint-auth-scheme-token",
    "description": "i18n:Token based endpoint authentication scheme",
    "type": "ms.vss-endpoint.service-endpoint-type",
    "targets": [
        "ms.vss-endpoint.endpoint-types"
    ],
    "properties": {
```

```
        "name": "Token",
        "displayName": "i18n:Token Based Authentication",
        "authenticationSchemes": [
            {
                "type": "ms.vss-endpoint.endpoint-auth-scheme-token",
                "headers": [
                    {
                        "name": "Authorization",
                        "value": "{{endpoint.apitoken}}"
                    }
                ],
                "inputDescriptors": [
                    {
                        "id": "apitoken",
                        "name": "i18n:API Token",
                        "description": "i18n:API Token for connection to
endpoint",
                        "inputMode": "textbox",
                        "isConfidential": true,
                        "validation": {
                            "isRequired": true,
                            "dataType": "string",
                            "maxLength": 300
                        }
                    }
                ]
            }
        ]
    }
}
```

Certificate-based authentication

This scheme takes one input - Certificate (confidential)

The value of certificate has to be provided in the text area.

JSON

```
{
    "id": "endpoint-auth-scheme-cert",
    "description": "i18n:Creates a certificate-based endpoint authentication
scheme",
    "type": "ms.vss-endpoint.service-endpoint-type",
    "targets": [
        "ms.vss-endpoint.endpoint-types"
    ],
    "properties": {
        "name": "Certificate",
        "displayName": "i18n:Certificate Based",
        "authenticationSchemes": [
```

```
            {
                "type": "ms.vss-endpoint.endpoint-auth-scheme-cert",
                "inputDescriptors": [
                    {
                        "id": "certificate",
                        "name": "i18n:Certificate",
                        "description": "Content of the certificate",
                        "inputMode": "TextArea",
                        "isConfidential": true,
                        "validation": {
                            "isRequired": true,
                            "dataType": "string"
                        }
                    }
                ]
            }
        ]
    }
}
```

No authentication

This scheme is used when an endpoint type doesn't require to take any input. For example, external services that support anonymous access to its resources.

JSON

```
{
    "id": "endpoint-auth-scheme-none",
    "description": "i18n:Creates an endpoint authentication scheme with no
authentication.",
    "type": "ms.vss-endpoint.endpoint-auth-scheme-none",
    "targets": [
        "ms.vss-endpoint.endpoint-auth-schemes"
    ],
    "properties": {
        "name": "None",
        "displayName": "i18n:No Authentication"
    }
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback

Static content hosting

Article • 10/04/2022

Azure DevOps Services

Choose to host static content for your extension, like HTML, CSS, and JavaScript files, on your own service, on a third-party hosting service, like Azure or Heroku, or on Azure DevOps Services directly.

> **Important**
>
> If your extension needs to create a custom table in the TFS database, do not create it using the '**dbo**' schema. Instead, custom tables should be created in a separate schema. For example, '**YourExtensionName**'.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Host on Azure DevOps Services

In this model, static content is packaged with your extension's .vsix file and is served from a public endpoint at `https://publisher.gallerycdn.vsassets.io`.

Your extension's static content is useful when you're enhancing or decorating data from Azure DevOps Services. The extension pub doesn't require you (the extension publisher) to set up, manage, or pay for hosting services for your extension

Steps

1. In your extension manifest file, specify the files you want to include via the `files` property:

```JSON
{
  "files": [
    {
        "path": "scripts", "addressable": true
```

```
    },
    {
        "path": "images/extra/icon1.png", "addressable": true
    }
  ]
}
```

2. Remove the `baseUri` property (if set) from your extension manifest.
3. Package your extension (steps)
4. Publish (or republish) your extension (steps)

> [!IMPORTANT] > Make sure to increment the version of your extension when you make changes to static content files included in your .vsix.

Keep in mind:

- The value specified by the `path` attribute can be a folder or individual file. If a folder, the entire folder (and any subfolders) is included.
- The `addressable` attribute is important and is what tells Visual Studio Codespaces to make the file(s) URL addressable.
- All `addressable` asset requests are case-sensitive. If the request for an asset has a different case than the actual uploaded asset, it results in an HTTP 404 (Not found) error.
- Not specifying a baseUri or setting an empty value tells Visual Studio Codespaces at runtime to calculate the base URI as if your static content's hosted by Azure DevOps Services.

Host on your own service (or a third-party service)

In this model, static content is served from your own service and not included in your extension's .vsix file.

Steps

1. Set the `baseUri` property in your extension manifest For example, assuming a value of `https://myservice.net/extension` and this hub contribution:

JSON

```
    "baseUri": "https://myservice.net/extension",
    "contributions": [
```

```
        {
            "id": "Fabrikam.HelloWorld",
            "type": "ms.vss-web.hub",
            "targets": [
                "ms.vss-work-web.work-hub-group"
            ],
            "properties": {
                "name": "Hello",
                "uri": "hello-world.html"
            }
        }
    ]
```

Azure DevOps Services loads the contents of this hub when it's rendered at `https://myservice.net/extension/hello-world.html`.

Next steps

Package, publish, and install extensions

Modal dialog

Article • 02/13/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The HostDialogService enables you to present a modal dialog to the user and prevent interaction with all parts of web access until the dialog is dismissed.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

In contrast to the standard dialog control, a modal dialog presented via the HostDialogService prevents interaction by the user on the entire page, not just within the extension.

Dialog contents

To start, declare a contribution of type `ms.vss-web.control` in your extension manifest. This contribution represents the content displayed within the dialog.

JSON

```
{
    "id": "registration-form",
    "type": "ms.vss-web.control",
    "description": "The content to be displayed in the dialog",
    "targets": [],
    "properties": {
        "uri": "registration-form.html"
    }
}
```

The `uri` property references a page that is rendered within the content area of the dialog:

HTML

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <script src="sdk/scripts/VSS.SDK.js"></script>
    </head>
    <body>
        <h2 id="header">Register now</h2>
        <p>
            <label>Name:</label>
            <input id="inpName" />
        </p>
        <p>
            <label>Date of birth:</label>
            <input id="inpDob" />
        </p>
        <p>
            <label>Email address:</label>
            <input id="inpEmail" />
        </p>
        <script>
            VSS.init();
            var registrationForm = (function() {
                var callbacks = [];

                function inputChanged() {
                    // Execute registered callbacks
                    for(var i = 0; i < callbacks.length; i++) {
                        callbacks[i](isValid());
                    }
                }

                function isValid() {
                    // Check whether form is valid or not
                    return !!(name.value) && !!(dateOfBirth.value) && !!
(email.value);
                }

                function getFormData() {
                    // Get form values
                    return {
                        name: name.value,
                        dateOfBirth: dateOfBirth.value,
                        email: email.value
                    };
                }

                var name = document.getElementById("inpName");
                var dateOfBirth = document.getElementById("inpDob");
                var email = document.getElementById("inpEmail");

                name.addEventListener("change", inputChanged);
                dateOfBirth.addEventListener("change", inputChanged);
                email.addEventListener("change", inputChanged);

                return {
                    isFormValid: function() {
                        return isValid();
```

```
                    },
                    getFormData: function() {
                        return getFormData();
                    },
                    attachFormChanged: function(cb) {
                         callbacks.push(cb);
                    }
                };
            })();

            // Register form object to be used across this extension
            VSS.register("registration-form", registrationForm);
        </script>
    </body>
</html>
```

Showing the dialog

To show the dialog (for example, when a user selects an action on a toolbar or menu), call the `openDialog` function on an instance of the HostDialogService, passing the fully-qualified identifer of the dialog content, for example `my-publisher.my-extension.registration-form` and any dialog options:

JavaScript

```
    VSS.getService(VSS.ServiceIds.Dialog).then(function(dialogService) {
        var extensionCtx = VSS.getExtensionContext();
        // Build absolute contribution ID for dialogContent
        var contributionId = extensionCtx.publisherId + "." +
extensionCtx.extensionId + ".registration-form";

        // Show dialog
        var dialogOptions = {
            title: "My Dialog",
            width: 800,
            height: 600
        };

        dialogService.openDialog(contributionId, dialogOptions);
    });
```

Showing the dialog (advanced)

A function can be called when the OK button is selected. This function is specified by `getDialogResult` in the options you provide when showing the dialog.

If a call to `getDialogResult` returns a non-null value, this value is then passed to the function specified by `okCallback` (also in the options) and the dialog is closed.

In this example, the `attachFormChanged` callback gets called when inputs on the form change. Based on the whether the form is valid or not, the OK button is enabled or disabled.

JavaScript

```
    VSS.getService(VSS.ServiceIds.Dialog).then(function(dialogService) {
        var registrationForm;
        var extensionCtx = VSS.getExtensionContext();
        var contributionId = extensionCtx.publisherId + "." +
extensionCtx.extensionId + ".registration-form";

        var dialogOptions = {
            title: "Registration Form",
            width: 800,
            height: 600,
            getDialogResult: function() {
                // Get the result from registrationForm object
                return registrationForm ? registrationForm.getFormData() :
null;
            },
            okCallback: function (result) {
                // Log the result to the console
                console.log(JSON.stringify(result));
            }
        };

        dialogService.openDialog(contributionId,
dialogOptions).then(function(dialog) {
            // Get registrationForm instance which is registered in
registrationFormContent.html
            dialog.getContributionInstance("registration-
form").then(function (registrationFormInstance) {

                // Keep a reference of registration form instance (to be
used previously in dialog options)
                registrationForm = registrationFormInstance;

                // Subscribe to form input changes and update the Ok enabled
state
                registrationForm.attachFormChanged(function(isValid) {
                    dialog.updateOkButton(isValid);
                });

                // Set the initial ok enabled state
                registrationForm.isFormValid().then(function (isValid) {
                    dialog.updateOkButton(isValid);
                });
            });
```

```
        });
    });
```

Enabling or disabling the OK button

Initially, the OK button is disabled. However, you can enable/disable this button by calling the `updateOkButton` method on the dialog:

JavaScript

```
    dialogService.openDialog(contributionId,
dialogOptions).then(function(dialog) {
        // Set true/false to enable/disable ok button
        dialog.updateOkButton(true);
    });
```

Passing values to the dialog control

It is possible pass initial values to dialog content when it is opened in the host dialog.

JSON

```
    {
        "id": "registration-form",
        "type": "ms.vss-web.control",
        "description": "The content displayed in the dialog",
        "targets": [],
        "properties": {
            "uri": "registration-form.html?id={{myId}}"
        }
    }
```

When the dialog is opened, following options need to be specified to pass `myId`:

JavaScript

```
    var dialogOptions = {
        title: "My Dialog Title",
        width: 800,
        height: 600,
        urlReplacementObject: { myId: new Date().getTime() }
    };
```

Customizing the dialog buttons

The `okText` and `cancelText` attributes can be used to specify alternate titles for the OK and Cancel buttons:

JavaScript

```
var dialogOptions = {
    title: "My Dialog Title",
    width: 800,
    height: 600,
    okText: "Yes",
    cancelText: "No"
};
```

To not show any buttons on the dialog, you can set the `buttons` attribute to `[]`:

JavaScript

```
var dialogOptions = {
    title: "My Dialog Title",
    width: 800,
    height: 600,
    buttons: []
};
```

More information

If you have a question or are looking for more information, consider going to one of the following areas:

- Azure DevOps on Stack Overflow
- Developer Community

Host page navigation

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The HostNavigationService provides APIs for interacting with the parent host frame, including refreshing it and accessing the hash of the URL:

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

Get the current hash value

JavaScript

```
    // Get navigation service

VSS.getService(VSS.ServiceIds.Navigation).then(function(navigationService) {
        // Get current hash value from host url
        navigationService.getHash().then(function (hash) {
            console.log("Host hash value: " + hash);
        });
    });
```

Get notified when the hash value changes

JavaScript

```
    // Get navigation service

VSS.getService(VSS.ServiceIds.Navigation).then(function(navigationService) {
        navigationService.onHashChanged(function (hash) {
            // Adding #something to the end of browser url executes this
handler with the hash value "something"
            console.log("Hash changed to : " + hash);
        });
    });
```

Change the hash value

Two methods are available for changing the hash value of the host page URL:

- `setHash` adds a new entry to the browser history
- `replaceHash` does **not** add a new entry to the browser history

JavaScript

```
    // Get navigation service

VSS.getService(VSS.ServiceIds.Navigation).then(function(navigationService) {
        // Adds a new entry to browser history
        navigationService.setHash("new-hash-value");
    });
```

Refresh the host page

Following code piece shows how host page can be reloaded.

JavaScript

```
    // Get navigation service

VSS.getService(VSS.ServiceIds.Navigation).then(function(navigationService) {
        // Reload whole page
        navigationService.reload();
    });
```

Basic styles for your widgets

Article • 02/20/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

We recommend you use the basic styles provided via the Widget SDK. Using these styles helps you quickly and easily create a widget that's consistent with the rest of the widgets on the dashboard. To use these styles, add the below line inside the `VSS.require` block in the JavaScript code for your widget.

> **Tip**
>
> Check out our newest documentation on extension development using the **Azure DevOps Extension SDK**.

```JavaScript
    WidgetHelpers.IncludeWidgetStyles();
```

This pulls a stylesheet by the name sdk-widget.css and include it in the iframe for your widget. It includes styles for font-family, font-size, margin and paddings for your widget. It also includes styles for headings (h1, h2, h3 etc.), links, and more.

Similarly, to use common styles in the widget configuration, include the line below inside the `VSS.require` block in the JavaScript code for your widget configuration.

```JavaScript
    WidgetHelpers.IncludeWidgetConfigurationStyles();
```

This pulls a stylesheet by the name sdk-widget-configuration.css and include it in the iframe for your widget configuration. It includes styles for font-family, font-size and styles for common form elements like input, textarea, and select.

> **Note**
>
> For these styles to apply to your widget, you need to add a "widget" class on the HTML element that contains your widget. All styles from the sdk-widgets.css are scoped to this class. Similarly, add a "widget-configuration" class on the HTML

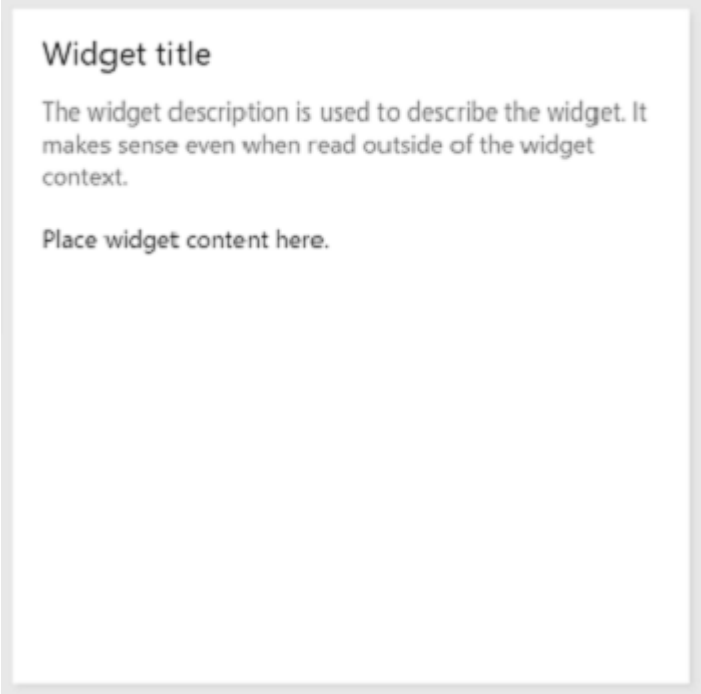
element that contains your widget configuration. All styles from the sdk-widget-configuration.css are scoped to this class.

Download the extension sample.

Widget body, title and description

By adding the class "widget" on the HTML element that contains your widget, you automatically get padding, font, and color for widget contents.

You should always have a title for your widget. This helps the user identify your widget and its functionality at a glance. Use `<h2>` with class "title". This also helps people using screen readers to quickly identify the different widgets on the dashboard.



> **Design principle:** Widgets should have a title. Use the `<h2>` tag with the "title" class.

Sometimes you might want to provide a small description about your widget or how to use it. In such cases, use the class "description" on the HTML element you wish to use for widget description.

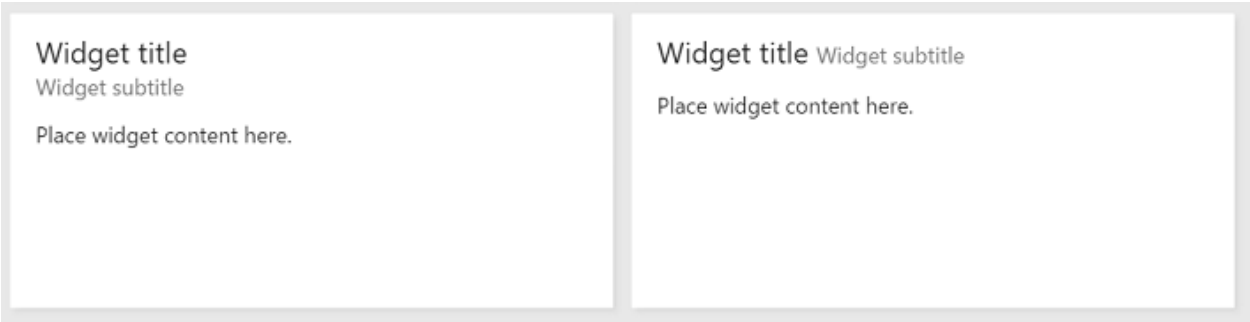
> **Design principle:** Use the "description" class for the widget description. Descriptions should make sense even when read out of the widget context.

HTML

```
    <div class="widget">
        <h2 class="title">Widget title</h2>
        <div class="description">The widget description is used to describe
the widget. It makes sense even when read outside of the widget context.
</div>
        <p>Place widget content here.</p>
    </div>
```

Widget titles and subtitles

Subtitles are text that supplement the title. They may not always make sense when read out of context without reading the title.



> **Design principle:** Use the "subtitle" class to provide more information about the widget. It may not make sense out of the widget context.

Use the below structure and classes "title", "inner-title" and "subtitle" to get the right font, color and margins for a title and subtitle combination. The title gets a greater font-size than the subtitle. The subtitle has a subdued color relative to the title or rest of the widget.

HTML

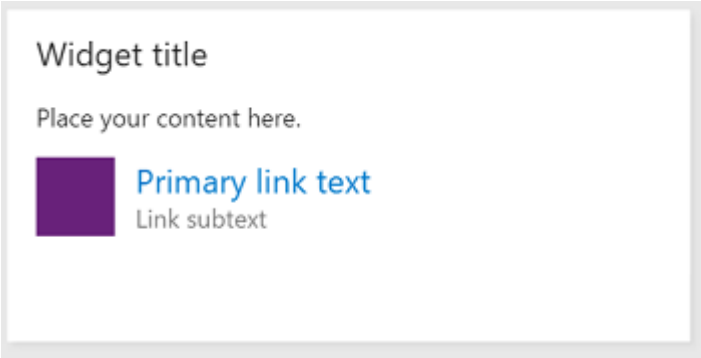
```
    <div class="widget">
        <h2 class="title">
            <div class="inner-title">Widget title</div>
            <div class="subtitle">Widget subtitle</div>
        </h2>
        <div class="content">
            Place widget content here.
        </div>
    </div>
```

You can use any html element for the title and subtitle combination. Here are some tips:

- When you need the subtitle to appear in the same line as the title, use an inline element like `<span>`
- When you need the subtitle to appear in the next line from the title, use a block element like `<div>`

Display hyperlinks, icons, text, and subtext in a widget

Some widgets have links which have an icon, text and subtext per link.



> **Design principle:** Use links with an icon and subtext to make the purpose of the link obvious to the user. Ensure that the icon symbolizes the link's target.

To get the same look and feel, use the below HTML structure and classes.

HTML

```
    <div class="widget">
        <h2 class="title">Widget title</h2>
        <div class="content">
            <p>Place your content here.</p>
            <a class="link-with-icon-text" href="http://bing.com"
target="_blank">
                <span class="icon-container" style="background-color:
#68217A"></span>
                <div class="title">
                    Primary link text
                    <div class="subtitle">Link subtext</div>
                </div>
            </a>
        </div>
    </div>
```

Display counters in a widget

The primary purpose of some widgets is to display the count of some data. The Query Tile and the Code Tile widgets are examples in this category of widgets. To use the same

styles as these widgets, add the "big-count" class on the HTML element holding the number to get the big font that is used by the Query Tile and the Code Tile widgets.

> **Design principle:** Use the "big-count" class to present the user with numbers in large font. It should not be used with non-numeric characters.

HTML

```
<div class="widget">
    <h2 class="title">Counter widget</h2>
    <div class="big-count">223</div>
    <div>Additional text</div>
</div>
```

Make a widget a hyperlink

Clicking anywhere on some widgets redirects the user to another page. To have your widget do the same, you can:

- Add an anchor tag as a child to the HTML element that acts as your widget container.
- Put all your widget content inside the anchor tag.
- Since your widget is hosted in an iframe, add the attribute "target" with value "_blank" to the anchor tag so that the link opens in a new tab/window instead of inside the same iframe.
- In addition to the "widget" class, add the "clickable" class to the widget container.

Your widget content gets the correct colors even though they are inside an anchor tag. Without the "clickable" class, the default blue color is forced on all text inside the widget. The widget also gets a custom visual cue on focus to help users who use the keyboard to navigate the dashboard.

> **Design principle:** Use the "clickable" class and the `<a>` tag to make the entire widget clickable. This is ideal when your widget is a summary of data available on

> another page.

HTML

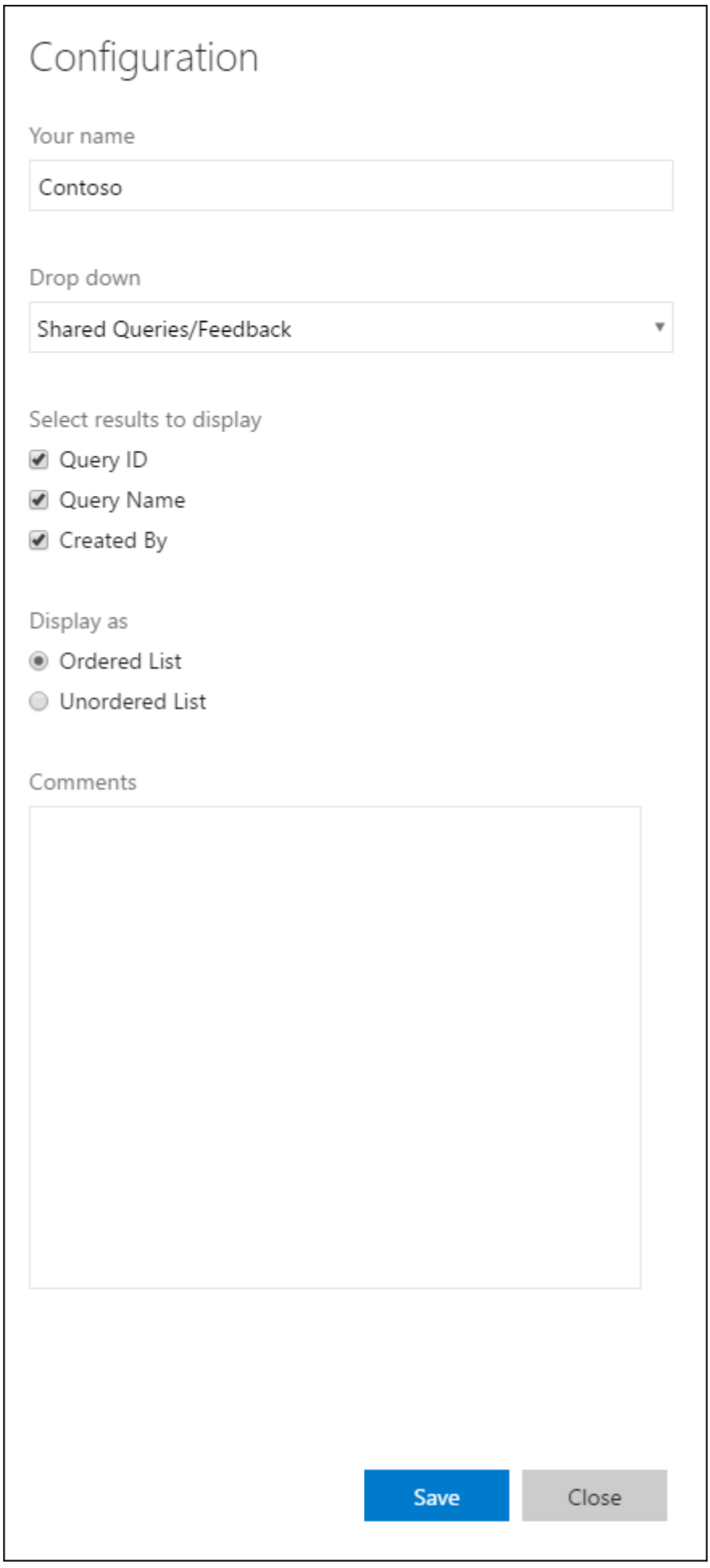
```
<div class="widget clickable">
    <a href="https://bing.com"  target="_blank">
        <h2 class="title">Counter widget</h2>
        <div class="big-count">223</div>
        <div>Click me!</div>
    </a>
</div>
```

Styles for common form elements in widget configuration

To use basic styles from the widget sdk for common form elements in widget configuration, follow these guidelines:

Form element	Wrapping element	Guidelines
Simple text box	`div` with class "single-line-text-input".	Use a `label` element to add text next to the text box. Use the `input` element to create a text box. Use the `placeholder` attribute to provide placeholder text.
Checkbox	`fieldset` with class "checkbox"	Use a `label` element to add text next to each checkbox. Use a `legend` element to caption the group of checkboxes. Use the `for` attribute on each `label` element to help screen readers understand the form element.
Radio button	`fieldset` with class "radio"	Use a `label` element to add text next to each radio button. Use a `legend` element to caption the group of radio buttons. Use the `for` attribute on each `label` element to help screen readers understand the form element.
Dropdown	`div` with class "dropdown"	Use a `label` element to add text next to the dropdown. If you want a dropdown occupying half the width, add class "half" to the wrapping `div` element. If you want to use the standard arrow icon from the sdk instead of the one provided by the browser, wrap the `select` element with another `div` with class "wrapper".
Multi-line text box	`div` with class "multi-line-text-input".	Use `label` element to label the `textarea` element used as multi-line text box.

The example below uses each of the form elements listed in the table.



HTML

```
<div class="widget-configuration">

    <div class="single-line-text-input" id="name-input">
        <label>Your name</label>
        <input type="text" value="Contoso"></input>
    </div>

    <div class="dropdown" id="query-path-dropdown">
        <label>Drop down</label>
        <div class="wrapper">
            <select>
                <option value="Shared Queries/Feedback">Shared
Queries/Feedback</option>
                <option value="Shared Queries/My Bugs">Shared Queries/My
Bugs</option>
                <option value="Shared Queries/My Tasks">Shared Queries/My
Tasks</option>
            </select>
        </div>

        <fieldset class="checkbox" id="select-results">
            <legend>Select results to display</legend>
            <input type="checkbox" id="check-option1" value="id"
name="check" checked="true">
            <label for="check-option1">Query ID</label><br/>
            <input type="checkbox" id="check-option2" value="name"
name="check" checked="true">
            <label for="check-option2">Query Name</label><br/>
            <input type="checkbox" id="check-option3" value="createdBy"
name="check" checked="true">
            <label for="check-option3">Created By</label><br/>
        </fieldset>

        <fieldset class="radio" id="display-options">
            <legend>Display as </legend>
            <input type="radio" id="radio-option1" value="ordered"
name="radio" checked="true">
            <label for="radio-option1">Ordered List</label><br/>
            <input type="radio" id="radio-option2" value="unordered"
name="radio">
            <label for="radio-option2">Unordered List</label><br/>
        </fieldset>

        <div class="multi-line-text-input">
            <label>Comments</label>
            <textarea></textarea>
        </div>
    </div>
</div>
```

Display validation errors below a form element

We recommend providing validation errors below the relevant form elements. To display these messages in a manner consistent with 1st party widgets, add the following code snippet under each form element for which you want to show the error message.

HTML

```
<span class="validation-error">
    <span class="icon-error-exclamation"></span>
    <span class="validation-error-text"></span>
</span>
```

The previous code snippet has the visibility hidden by default. Whenever you want to display an error message, find the corresponding "validation-error-text", add text to it and set `visibility:visible` on its parent.

Example: There is a simple text box where the user needs to type in a string. You need to show an error message if the text box is empty.

The html for this would be:

HTML

```
<div class="widget-configuration">
    <div class="single-line-text-input">
        <label>Your name</label>
        <input type="text">Type Here</input>

        <span class="validation-error">
            <span class="icon-error-exclamation"></span>
            <span class="validation-error-text"></span>
        </span>
    </div>
</div>
```

And the JavaScript code behind this would be:

JavaScript

```
var $singleLineInput = $(".single-line-text-input input");
var $errorSingleLineInput = $(".single-line-text-input input .validation-
```

```
error-text");

$singleLineInput.on("input", function(){
        if ($singleLineInput.val() == ""){
            $errorSingleLineInput.text("Please enter your name.");
            $errorSingleLineInput.parent().css("visibility", "visible");
            return;
        }
        $errorSingleLineInput.parent().css("visibility", "hidden");
    });
```

Deploy web content to Azure

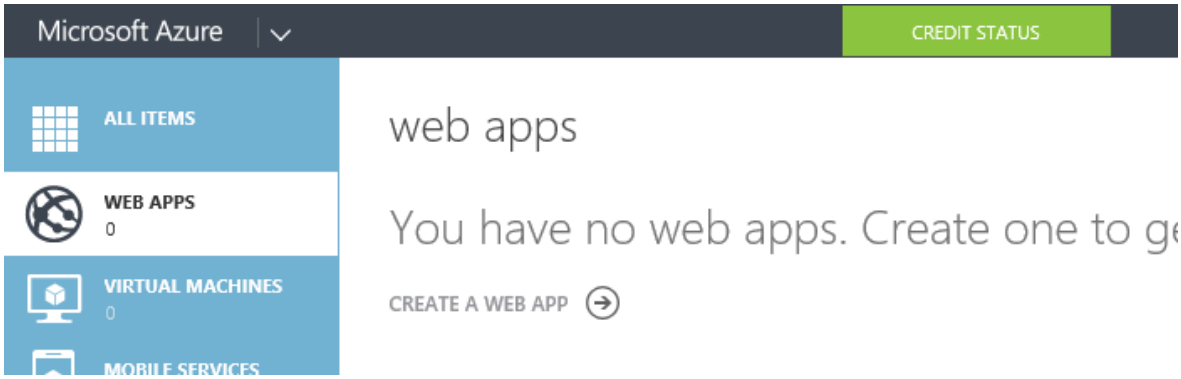
Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

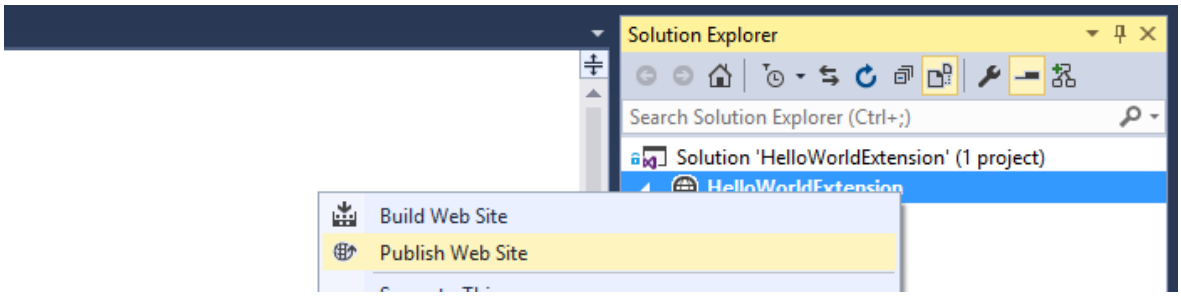
> If your extension doesn't require any back-end logic, consider hosting your static content directly on Azure DevOps. See **content hosting**.

After you've created an extension, you can publish it to Azure so that it's available in the cloud.

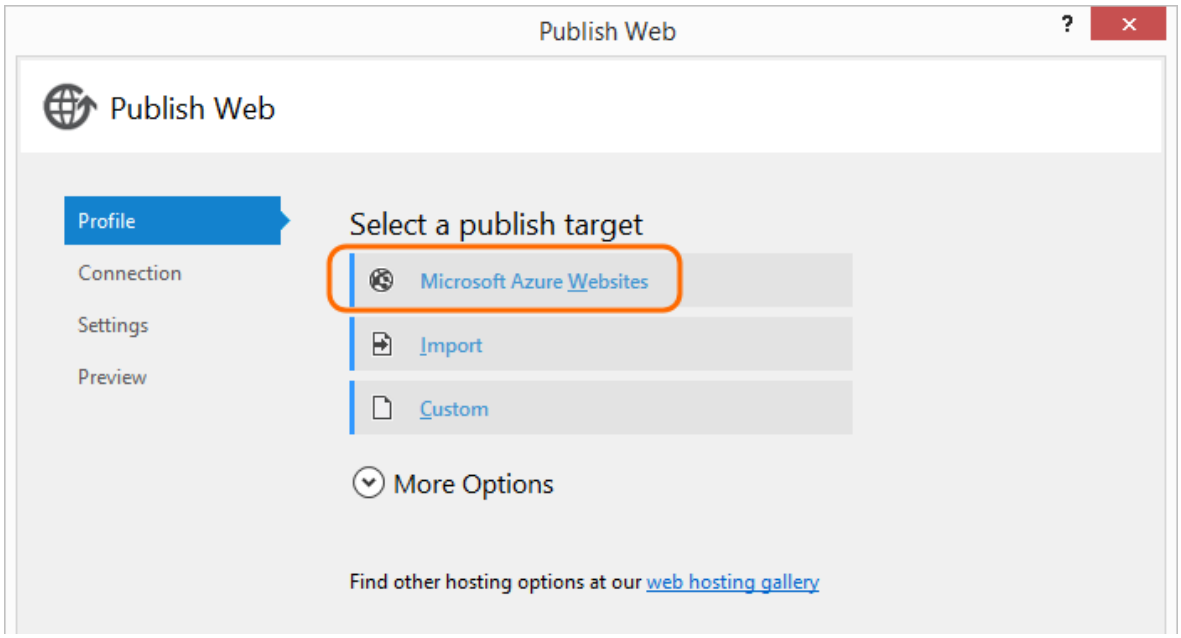
1. If you don't have an Azure subscription, create one. You can use the free trial.
2. Create a web app in Microsoft Azure to host your extension.



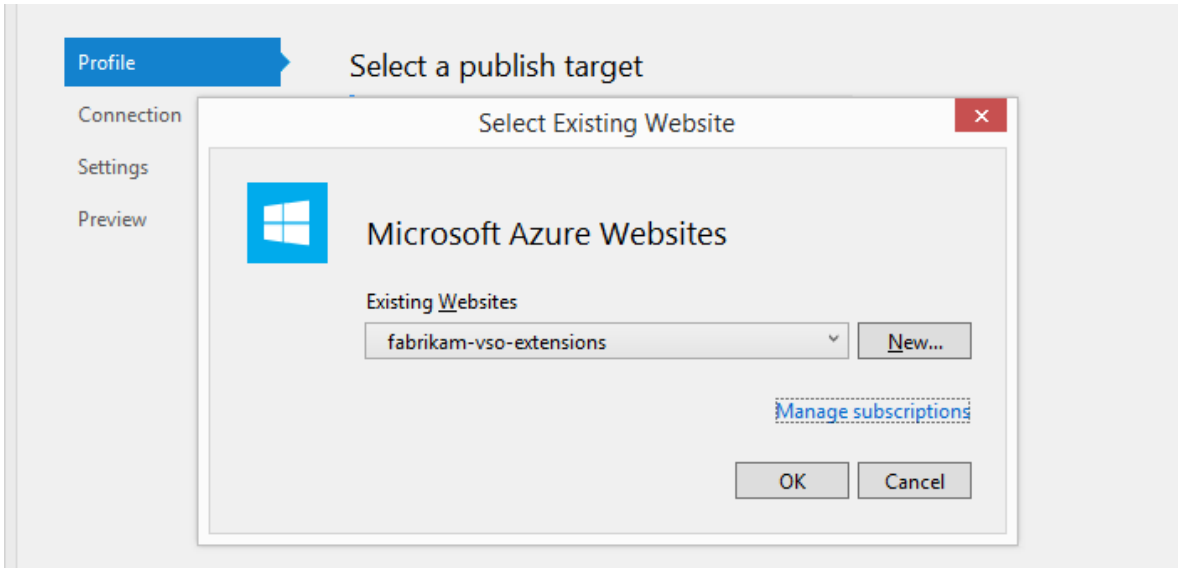
3. Publish your web site from the solution explorer.



4. Publish to Azure.

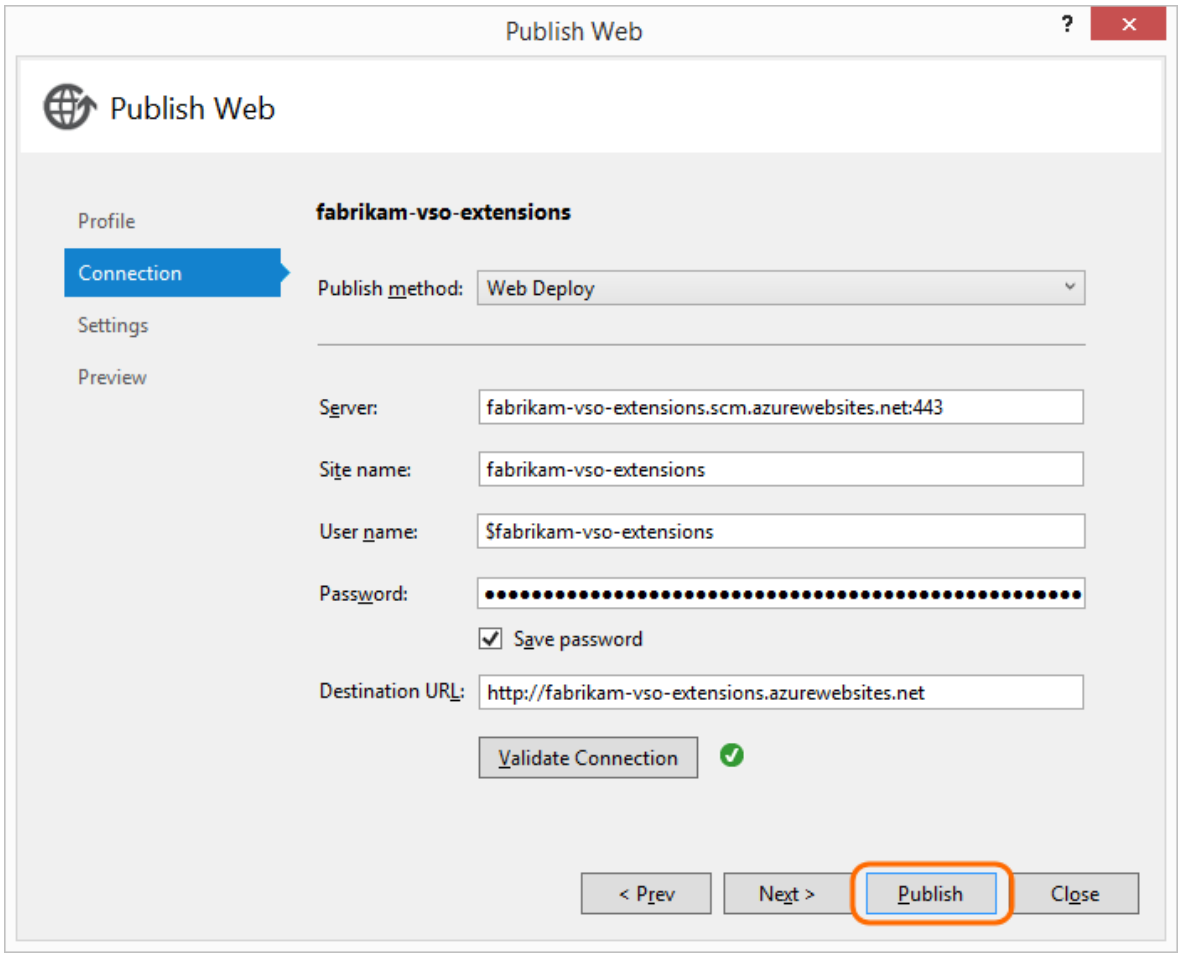


5. Pick the web app that you set up to host your extension.



If your web site doesn't show up, use the **Manage subscriptions** dialog to connect your Visual Studio organization to your Microsoft Azure subscription.

6. Publish your extension.



7. Change your extension manifest to use your Microsoft Azure web app instead of localhost.

```
JSON
```

```
"baseUri": "https://fabrikam-vso-extensions.azurewebsites.net/",
```

8. Install your extension again and try it out.

Integrate with service hooks

Article • 07/19/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

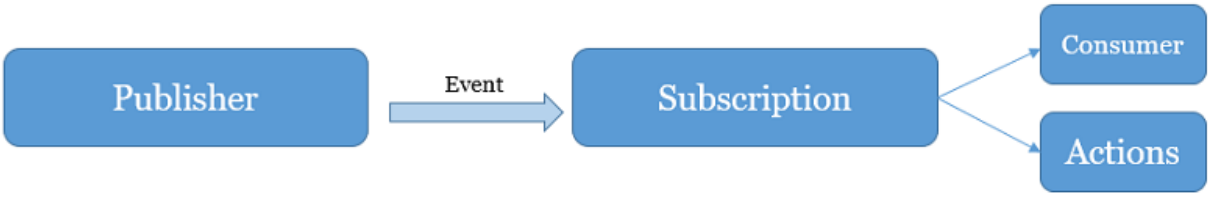
Service Hooks let you run tasks on other services when events happen in your project in Azure DevOps.

For example, you can create a card in Trello when a work item gets created or send a push notification to your team's mobile devices when a build fails. You can also use service hooks in custom apps and services as a more efficient way to drive activities when events happen in your projects.

How do service hooks work?

Service hook **publishers** define a set of *events* that you can subscribe to. **Subscriptions** listen for these *events* and define **actions** to take based on the event.

Subscriptions also target **consumers**, which are external services that can run their own actions when events occur.



> **Note**
>
> Service hooks require allowing IP ranges for inbound connection to service endpoints. The service endpoint is the set of properties provided to a service hook. For more information, see Allowed address lists and network connections, IP addresses and range restrictions.

Available services

The following services are available as the target of service hooks. For more information about other apps and services that integrate with Azure DevOps, go to the Visual Studio Marketplace.

Expand table

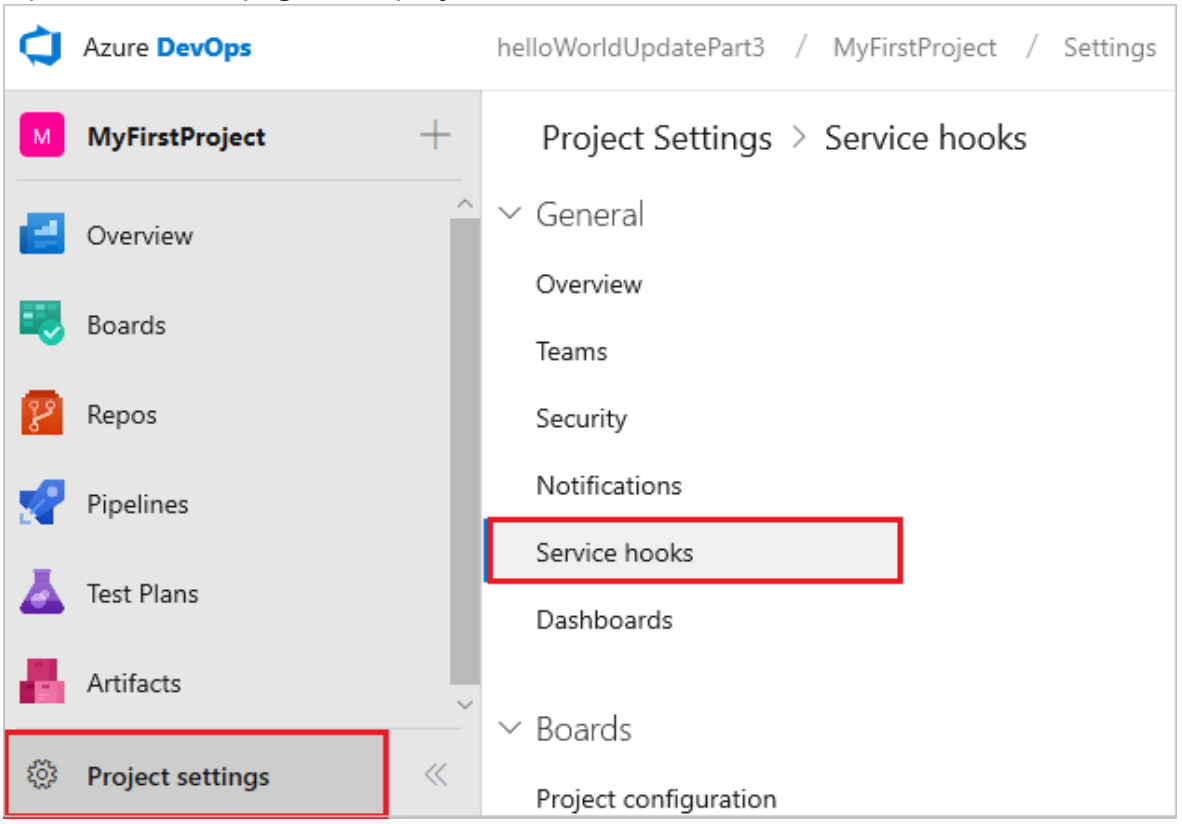
Service	Supported events	Supported actions
App Center	Work item updated	Send notification. Subscriptions are managed through App Center.
AppVeyor	Code pushed	Trigger an AppVeyor build. Subscriptions are managed through AppVeyor.
Azuqua	All	Post event to FLO. Subscriptions are managed through Azuqua.
Azure App Service	Code pushed	Deploy web app. Subscriptions are managed through Azure App Service.
Azure Service Bus	All	Send a message to a Notification Hub, Service Bus Queue, or Service Bus Topic. Subscription is managed through Azure App Service.
Azure Storage	All	Insert a message in a Storage Queue.
Bamboo	Build completed, Code pushed	Queue a build.
Datadog	All	Post an event in Datadog.
Grafana	Release deployment completed	Add annotation to Grafana database.
Jenkins	Build completed, code pushed, PR merge attempted, release deployment completed	Trigger generic or Git build.
Microsoft Teams	All	Post a message to a channel. Subscriptions are managed in Microsoft Teams.
MyGet	Build completed, code pushed	Publish NuGet package to MyGet. Trigger a MyGet build. Subscriptions are managed by MyGet.
Office 365	All	Post a message to a group.
Slack	All	Post a message to a channel.
Trello	All	Create a card or list.
UserVoice	Work item created, work item updated	Send linked work item event. Subscriptions are managed through UserVoice.
Web Hooks	All	Post via HTTP.

Service	Supported events	Supported actions
Workplace Message Apps	All	Send notifications. Subscriptions are managed through Workplace Message Apps.
Zapier	All	Send notification. Subscriptions are managed through Zapier.
Zendesk	Work item commented on	Create a private comment in a ticket.

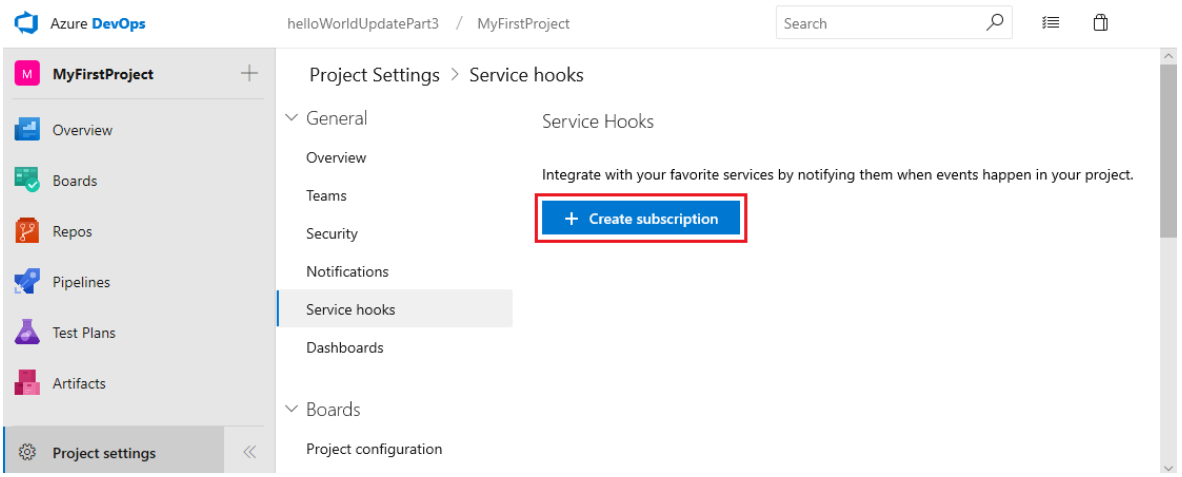
Create a subscription

When you integrate one of these services with Azure DevOps, you must create a new subscription. In many cases, you need to do some configuration in the other service, too. For specific details, check out the information for the service that you're interested in.

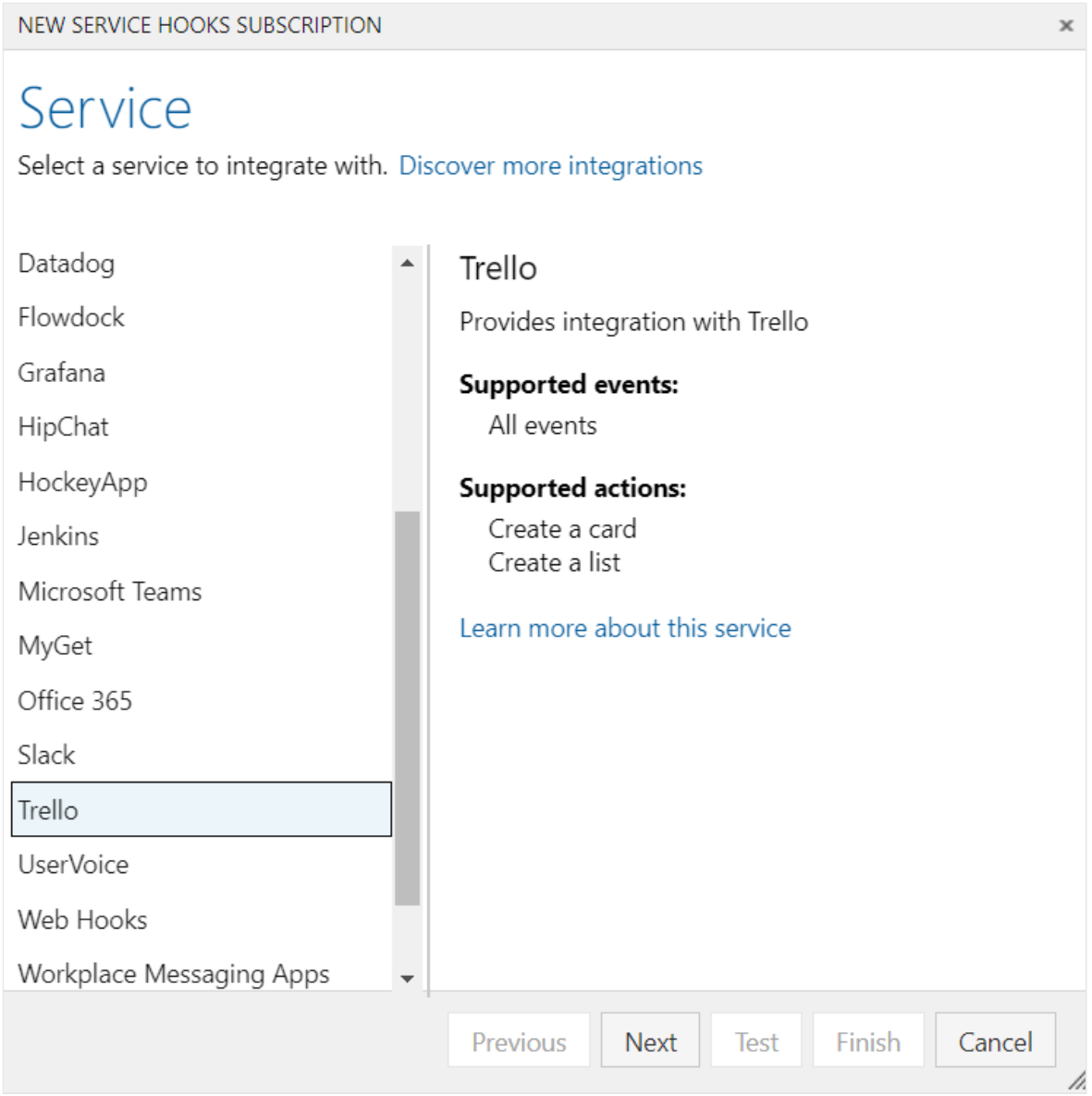
1. Open the admin page for a project in web access.



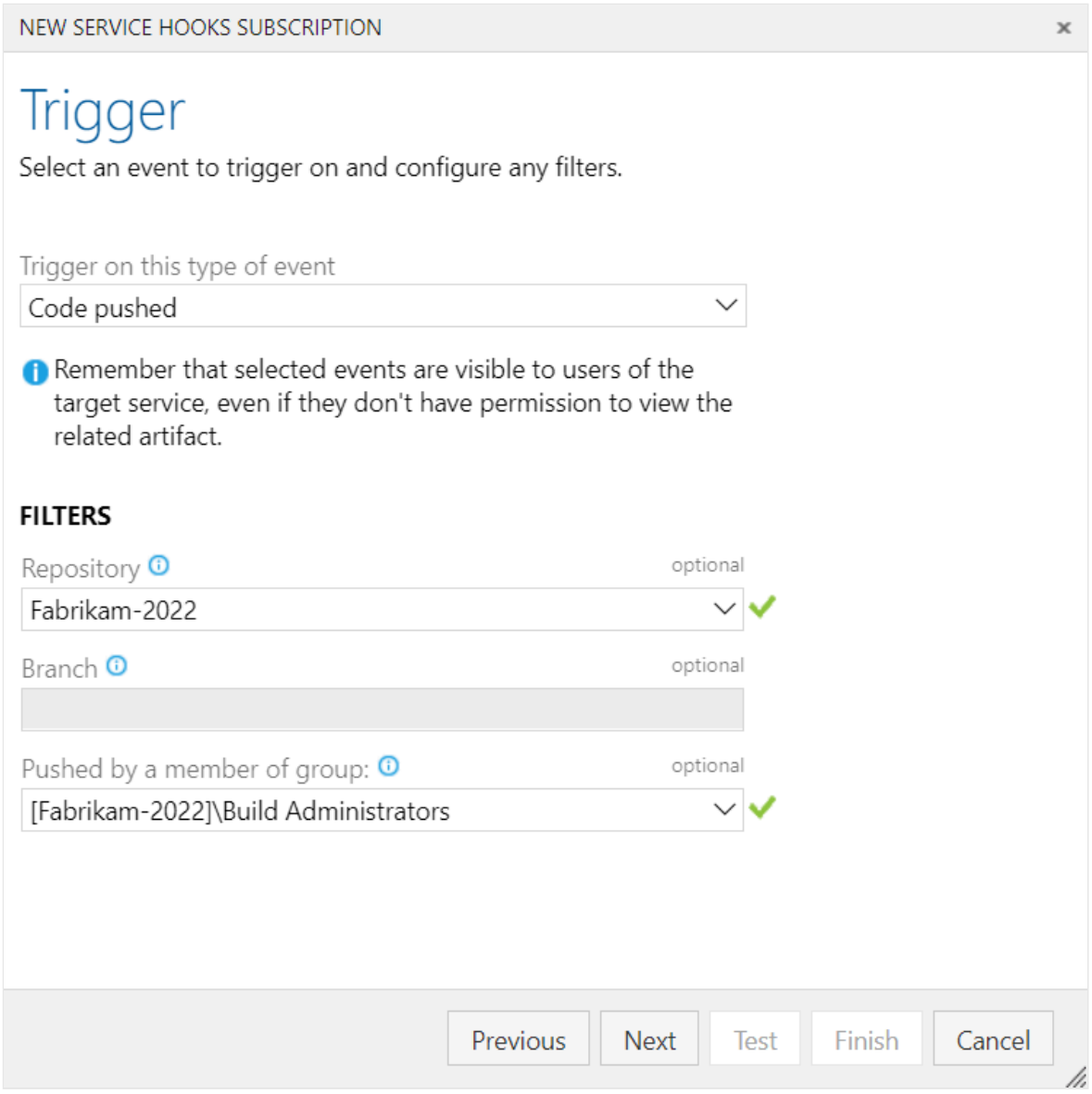
2. Run the wizard to create the subscription.



3. Select the service that you want to integrate with.

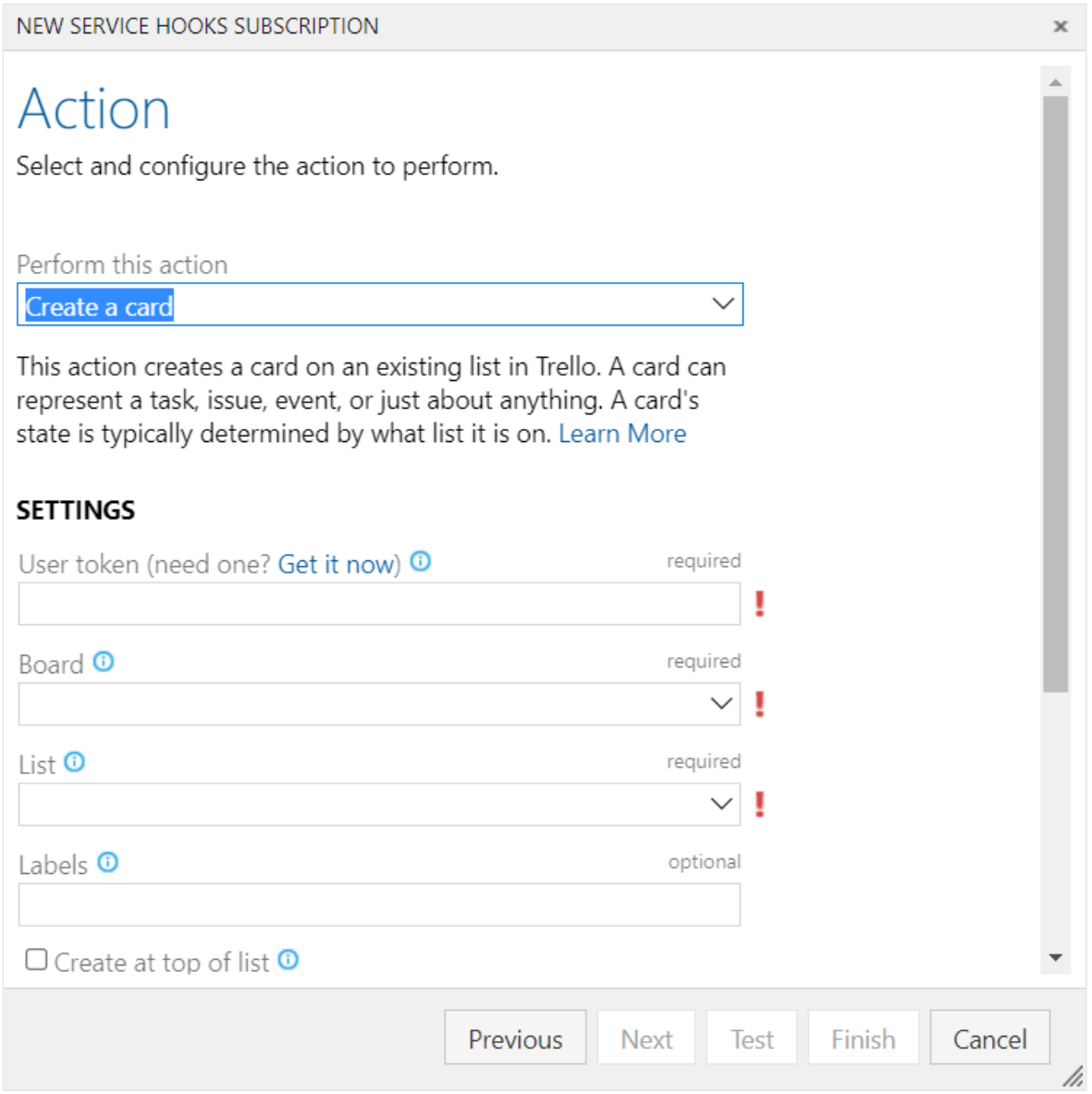


4. Select the event to trigger on and any applicable filters.

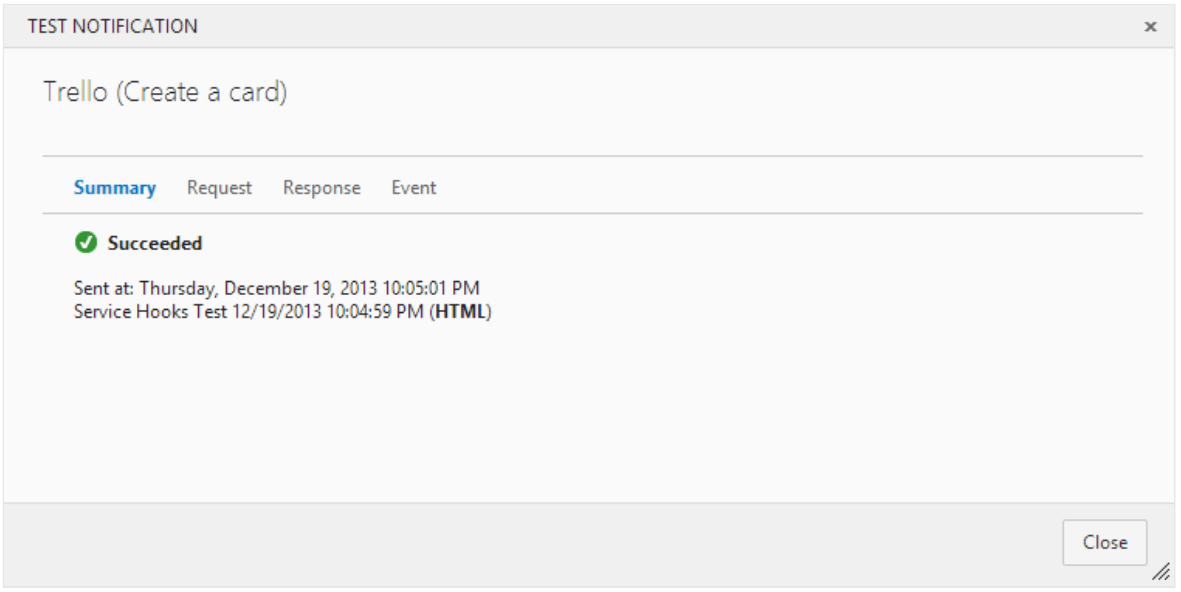


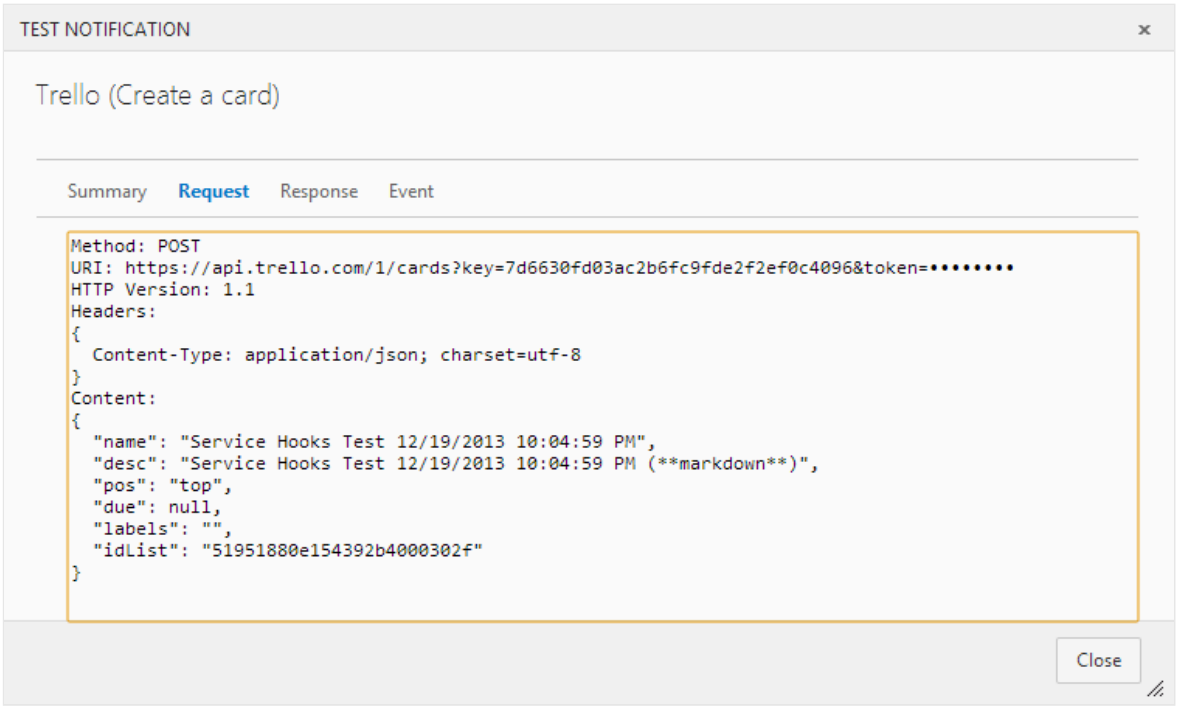
5. Select an action to run on the target service.

> **Note**
>
> The list of available actions may be limited based on the event type that you selected.



6. To confirm the settings are correct, test the subscription and then finish the wizard.





FAQs

Q: What permissions do I need to set up a subscription?

A: *Edit subscriptions* and *View subscriptions*. By default, only project administrators have these permissions. To grant them to other users directly, you can use the command line tool or the Security REST API.

To grant *View* permission for a group, see Set *View* permission for a group in Service Hooks page.

Q: What are the security implications of granting *Edit subscriptions* and *View subscriptions* permissions?

A: The user with these permissions can see all subscriptions created in the project and the notification history for those subscriptions. That user can then create any type of service hook subscription in that project. If the user sets up a subscription for a resource that they don't otherwise have permission to access, the subscription won't be triggered.

For example: Subscriptions you create for work item update events in area path XYZ won't send notifications if you lack access to those work items. However, you can see the notification history of other users who do have access to those work items.

Q: Can I create service hook subscriptions for a project programmatically?

A: Yes. For more information, see Create a service hooks subscription.

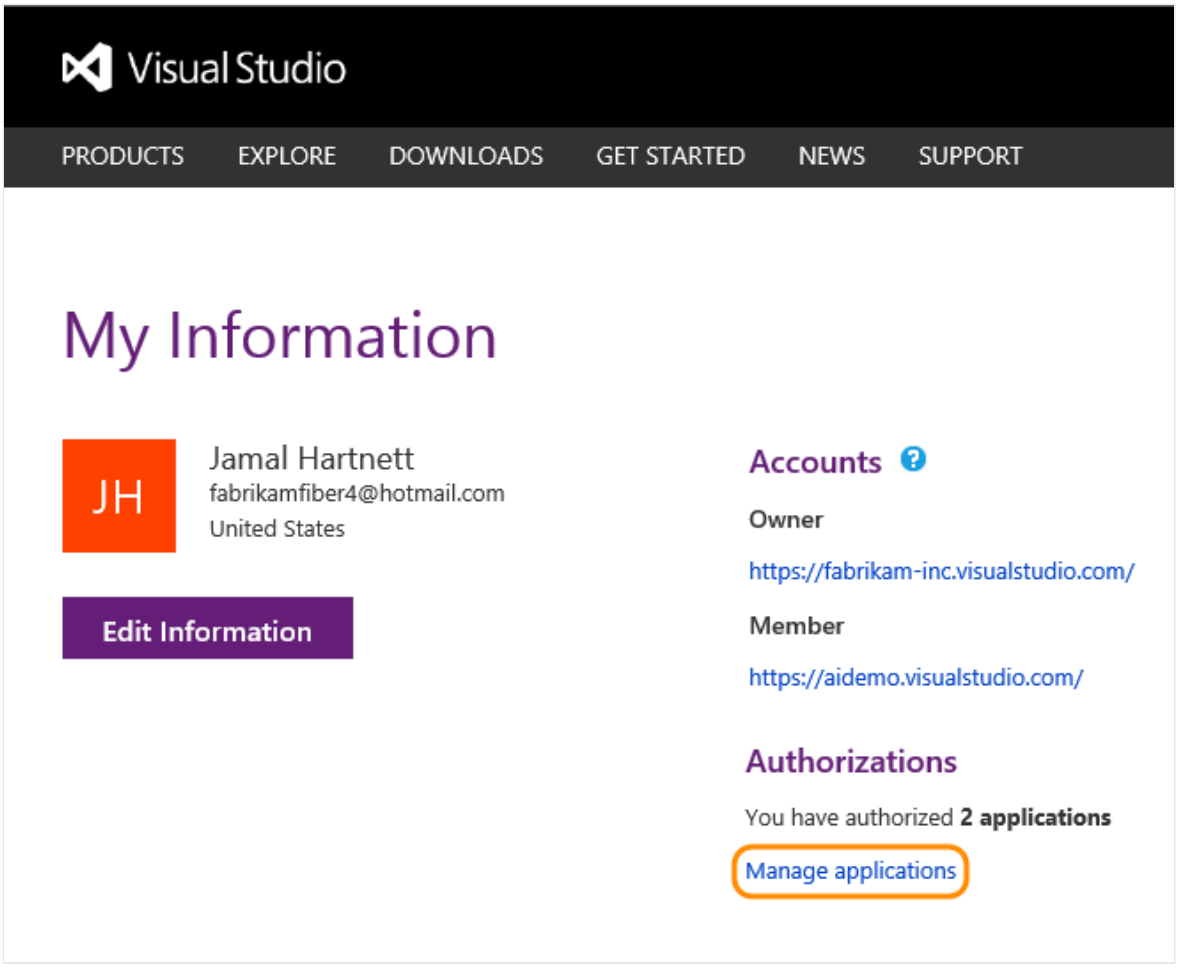
Q: Can I remove an app's access to my organization after I've authorized it?

A: Yes. You can revoke authorizations from your profile.

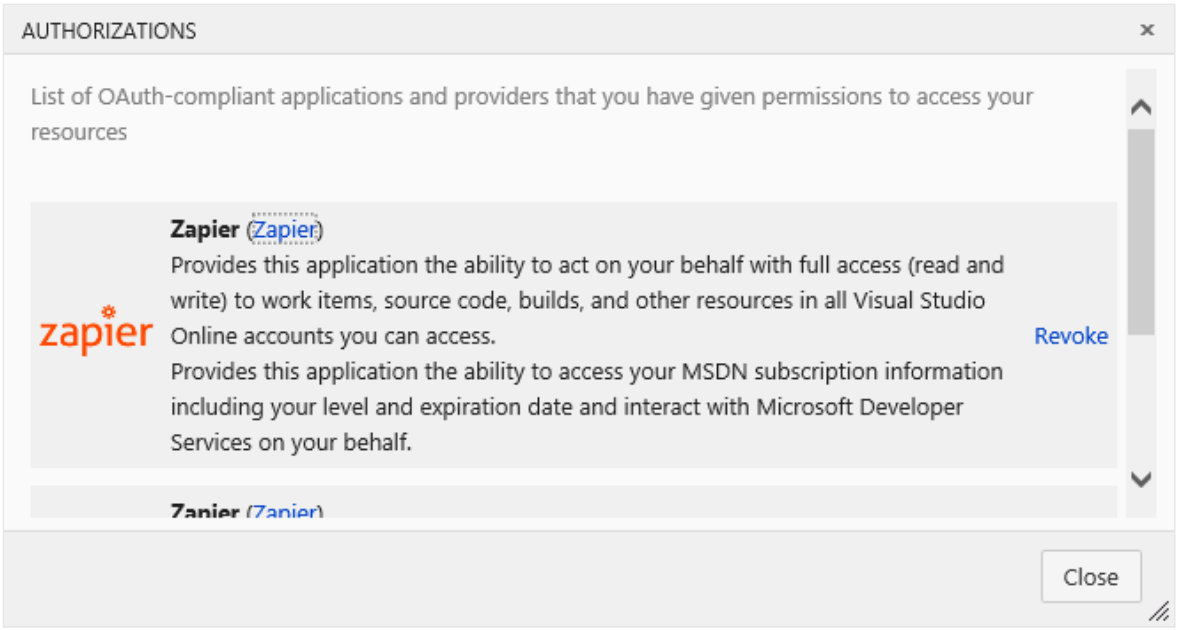
1. Go to your profile page from https://visualstudio.microsoft.com/ .

   Make sure you start from the Visual Studio site (https://visualstudio.microsoft.com/) instead of your organization (`https://dev.azure.com/{orgName}`) because your profile accessed from your organization takes you to the wrong implementation of the authorizations management feature.

2. Manage your authorizations.



3. Revoke any authorizations you no longer want to allow.



Q: Why can't we set up service hooks for HipChat anymore?

A: Atlassian officially dropped support for HipChat. For more information, see the announcement.

Related articles

- Troubleshoot service hooks and FAQs
- Visual Studio Marketplace
- Billing information

Feedback

Was this page helpful? Yes No

Provide product feedback

Workplace messaging apps

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The following integrations help users receive notifications in response to events in Azure DevOps within their workplace messaging apps such as Microsoft Teams and Slack.

Along with receiving notifications, below integrations also help users to complete workflows on Azure DevOps, such as allowing users to approve release deployments and creating work items from their channels. You can use these apps only with a project hosted on Azure DevOps Services.

Microsoft Teams

- Azure Boards app for Microsoft Teams
- Azure Pipelines app for Microsoft Teams
- Azure Repos app for Microsoft Teams

Slack

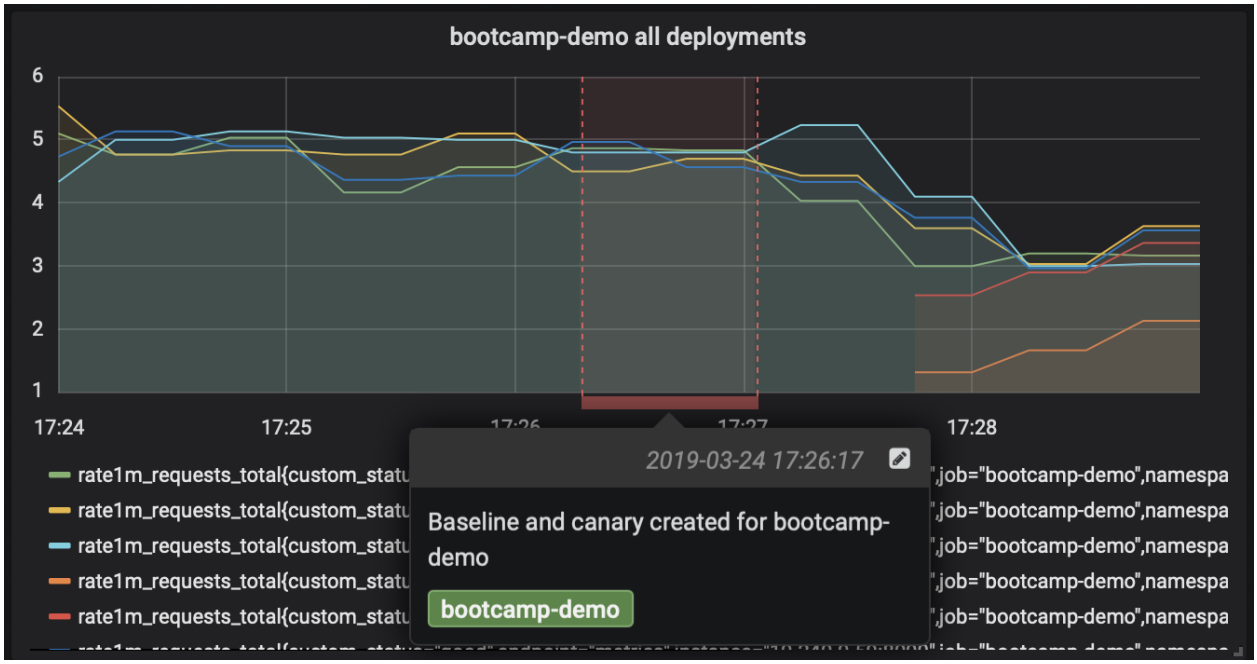
- Azure Boards app for Slack
- Azure Pipelines app for Slack
- Azure Repos app for Slack

Create a service hook for Azure DevOps with Grafana

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

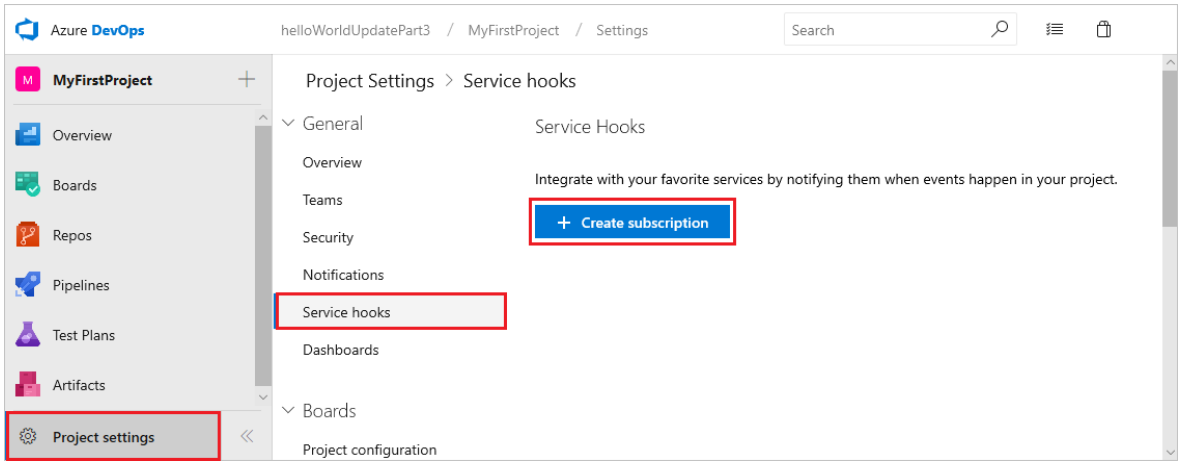
Grafana is an open source, feature-rich metrics dashboard and graph editor. It supports Graphite, Elasticsearch, OpenTSDB, Prometheus, and InfluxDB analytics services. Use the Grafana service hook to annotate Grafana dashboards upon completion of Azure Pipelines deployments.



Create a service hook subscription

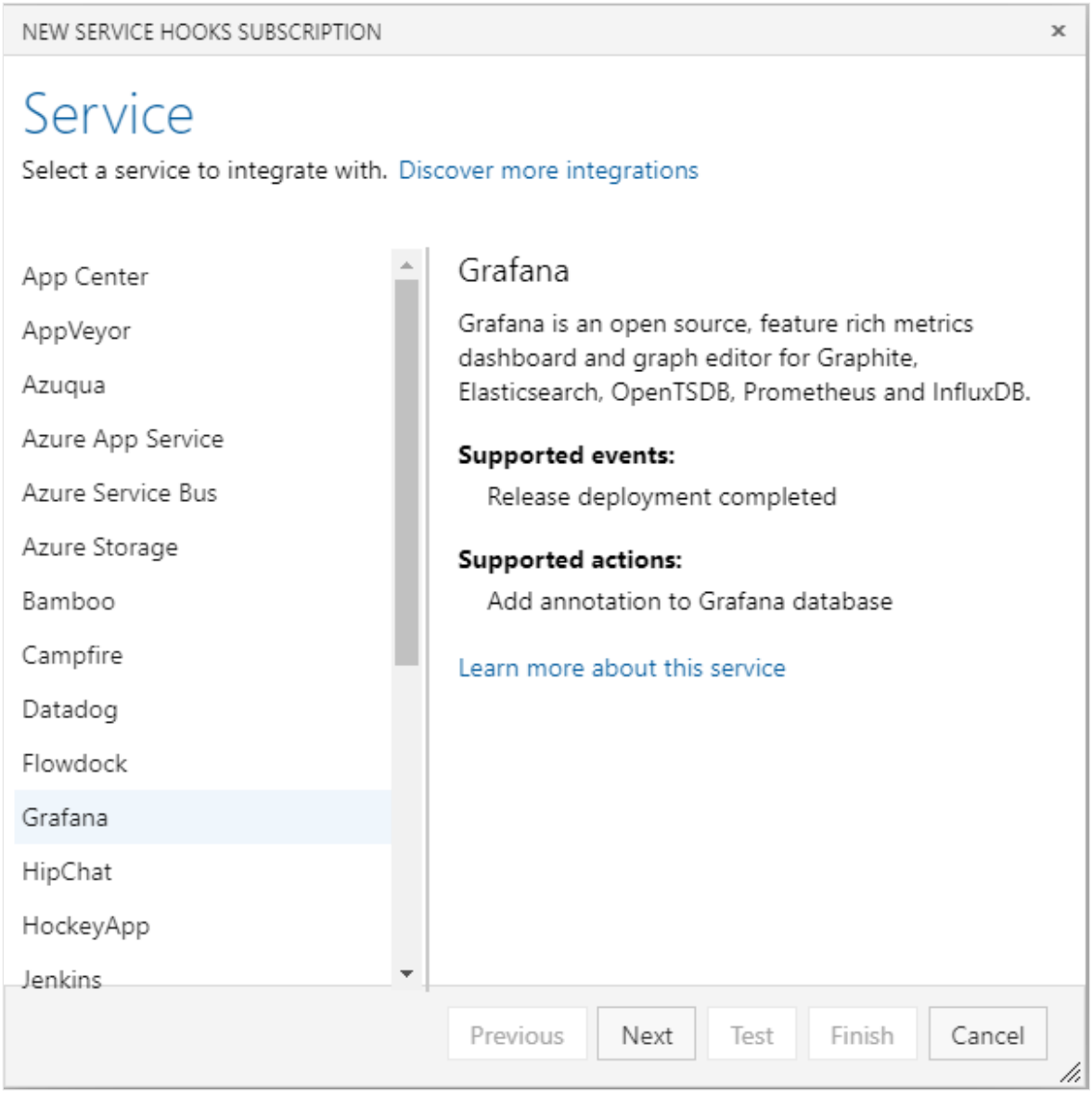
1. Navigate to project settings:

   `https://dev.azure.com/{orgName}/{project_name}/_settings/serviceHooks`

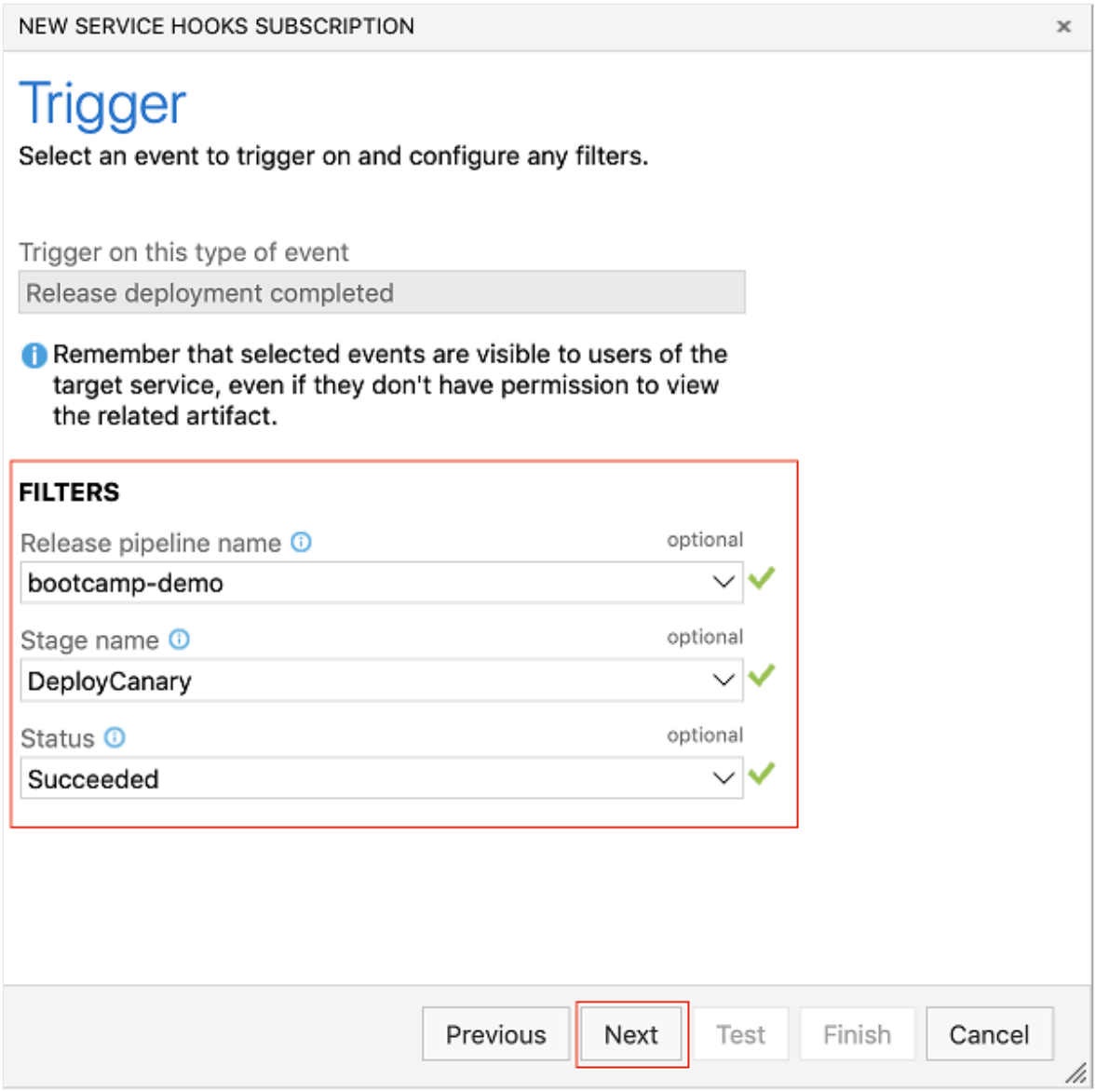


Select **Create Subscription**.

2. Choose **Grafana** among the list of services and choose **Next**



3. For **Release deployment completed** event, configure the optional filters: **Release pipeline name**, **Stage name**, and **Status**

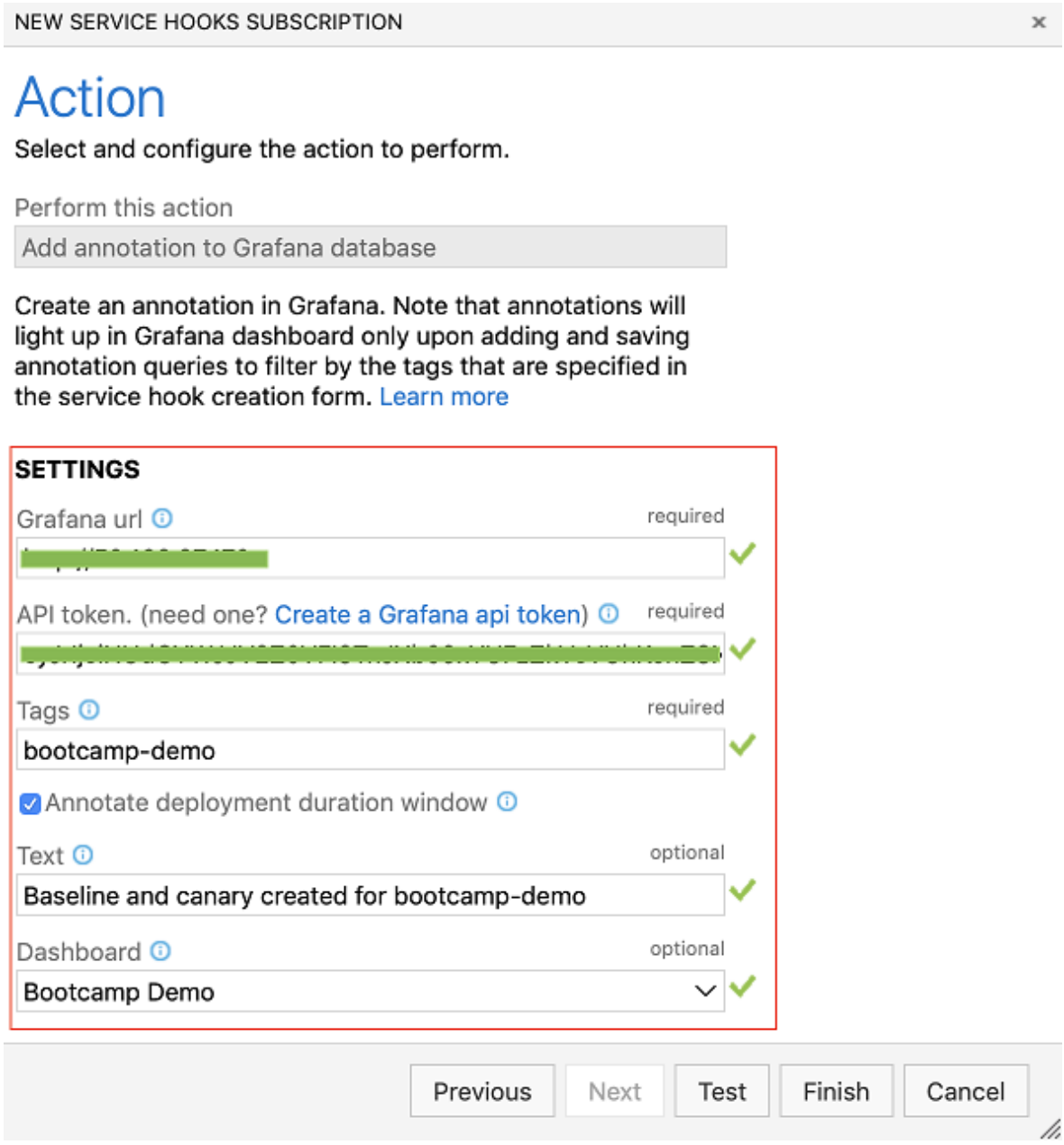


4. Provide the Grafana URL and Grafana API token required for Azure DevOps to post annotations.

    - If you check the **Annotate deployment duration window**, the annotation added corresponds to the duration of the deployment (start and end timestamps of deployment).
    - If not checked, annotation corresponds to only the completion timestamp of the deployment. The dashboard input can be used to choose a specific dashboard as the target of annotations rather than the default behavior of annotating all dashboards.

5. Choose **Test** to verify that Azure DevOps can use the provided configuration and successfully create a subscription.

6. Once verified, choose **Finish** to complete the creation of subscription.


NEW SERVICE HOOKS SUBSCRIPTION
Action
Select and configure the action to perform.
Perform this action
Add annotation to Grafana database
Create an annotation in Grafana. Note that annotations will light up in Grafana dashboard only upon adding and saving annotation queries to filter by the tags that are specified in the service hook creation form. Learn more
SETTINGS
Grafana url
required
API token. (need one? Create a Grafana api token)
required
Tags
required
bootcamp-demo
Annotate deployment duration window
Text
optional
Baseline and canary created for bootcamp-demo
Dashboard
optional
Bootcamp Demo
Previous
Next
Test
Finish
Cancel

Create a service hook for Azure DevOps with Slack

Article • 03/25/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

In this article, learn how to post messages to Slack in response to events in your Azure DevOps organization, such as completed builds, code changes, pull requests, releases, work items changes, and more.

> **Note**
>
> For Azure DevOps Services, we recommend you use the following suite of apps which offer features to integrate with Slack.

Azure Boards app for Slack

Azure Boards app for Slack helps to easily create and monitor work items from your Slack channels. You can create work items using a slash command, or use message actions to convert conversations in the channel into work items. You can also set up and manage subscriptions to get notifications in their channel whenever work items are created or updated.

Azure Pipelines app for Slack

The Azure Pipelines app for Slack helps to easily monitor the events in your pipelines. You can set up and manage subscriptions for completed builds, releases, pending approvals and more from the app and get notifications for these events in their channels. You can also approve release deployments from your channels.

Azure Repos app for Slack

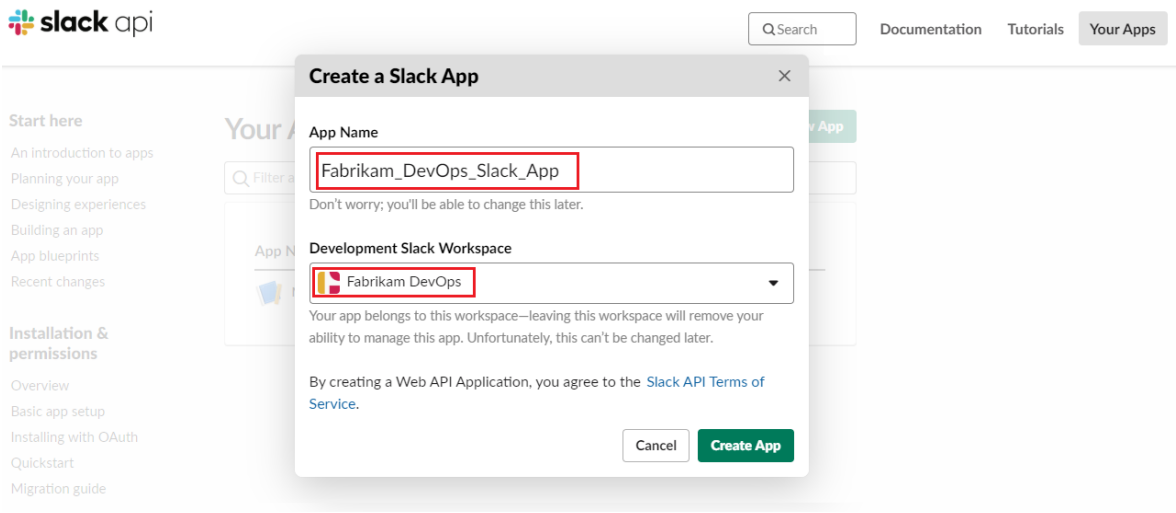
The Azure Repos app for Slack helps to easily monitor the events in your repositories. You can set up and manage subscriptions for code commits, PR creation and updates, and more, and get notifications for these events in their channels.

Prerequisites

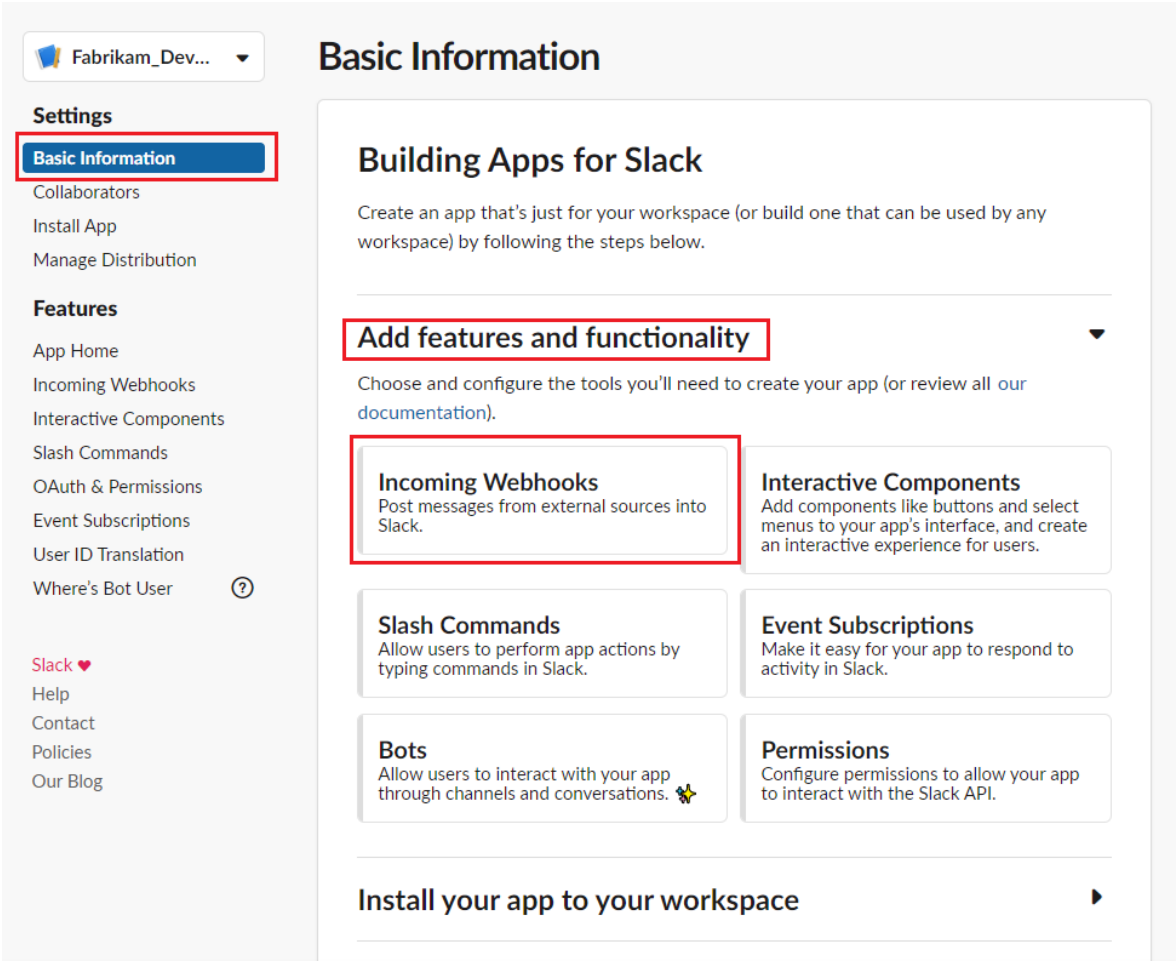
Refer to the Slack documentation, Sending messages using Incoming Webhooks to understand the process of using Web Hooks to push information to a Slack channel.

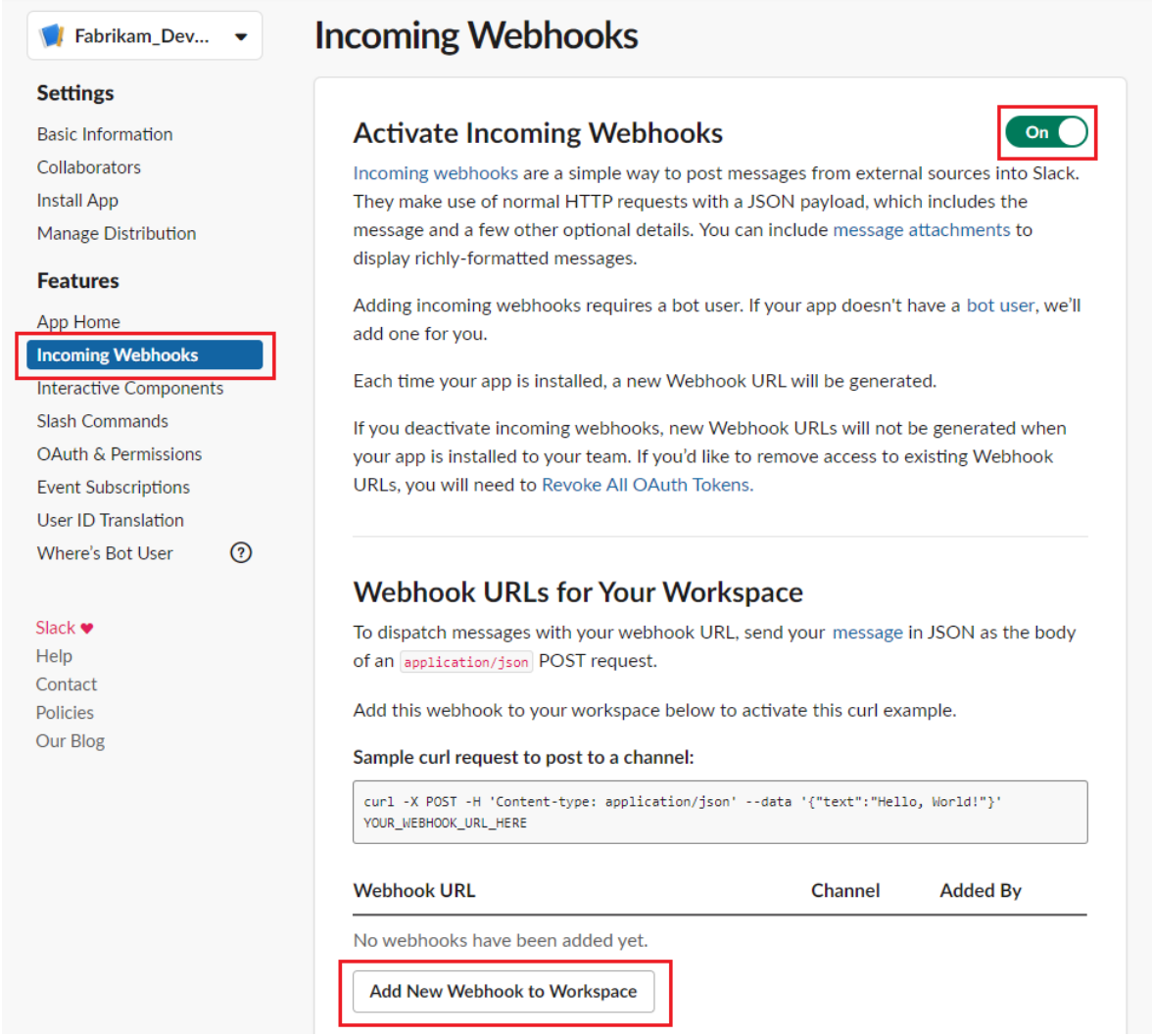
Create a custom app in Slack

1. Create a new Slack app

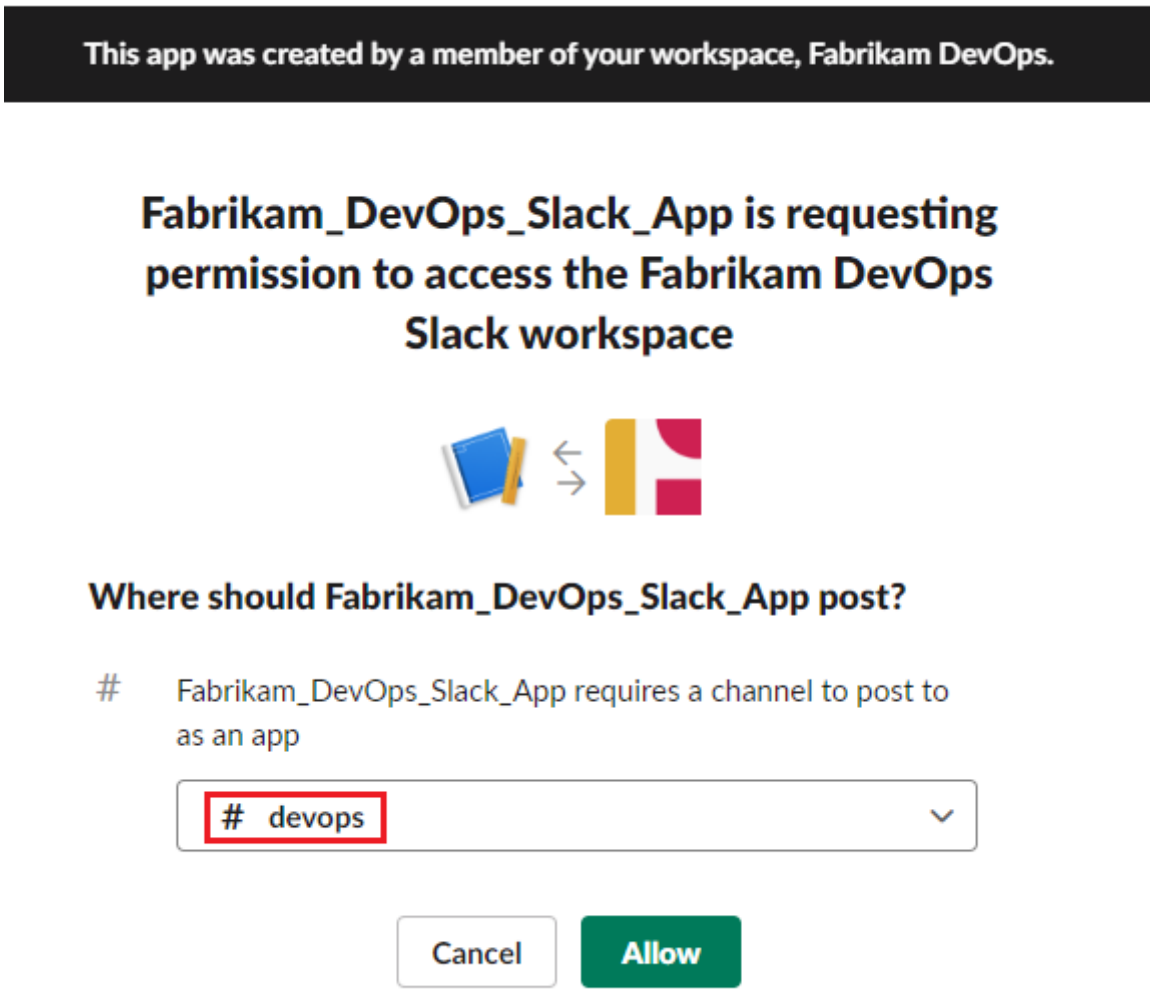


2. Activate incoming Web Hook and add a new Web Hook to the desired workspace

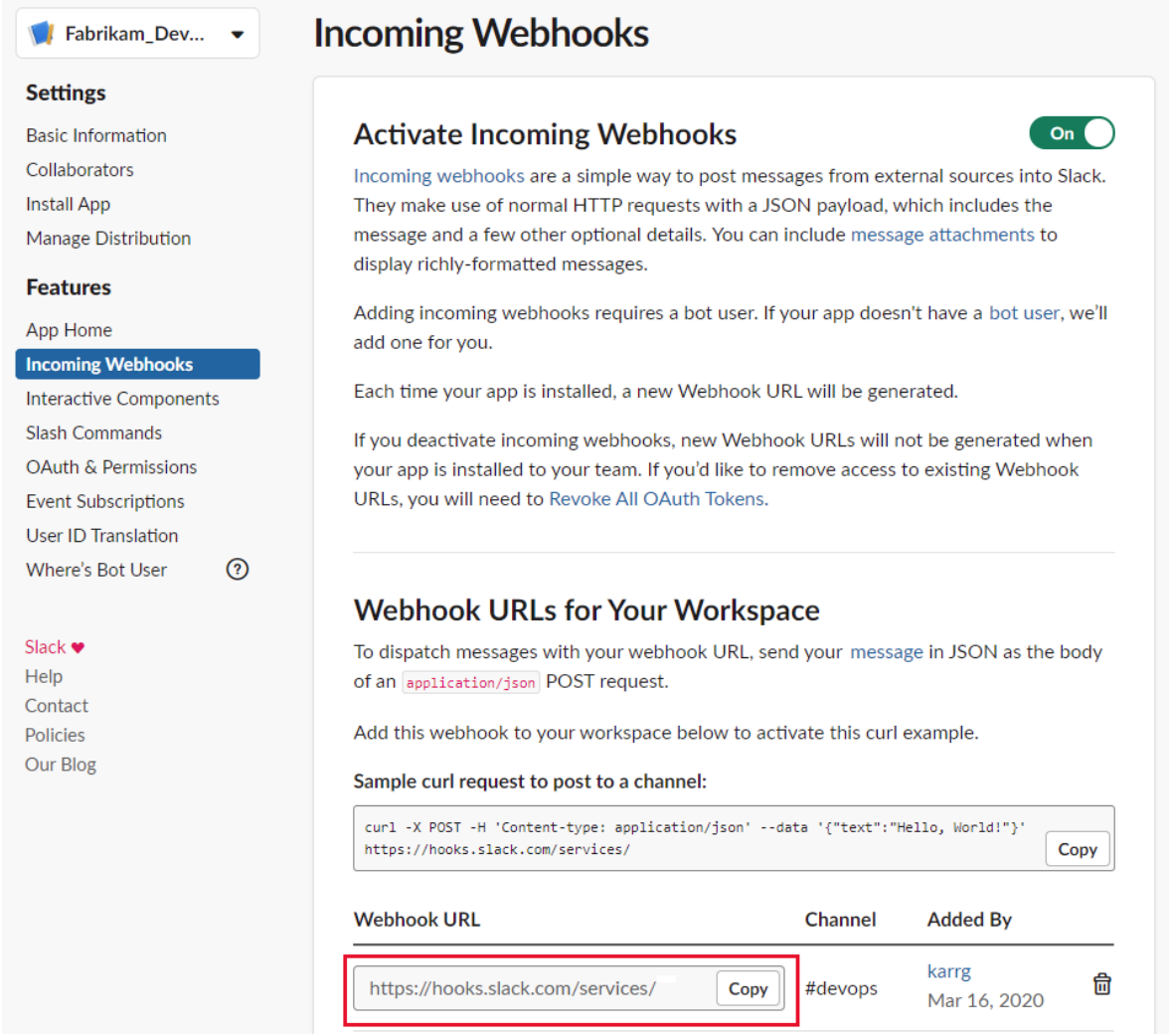




3. Select the channel for which Web Hook must be created.



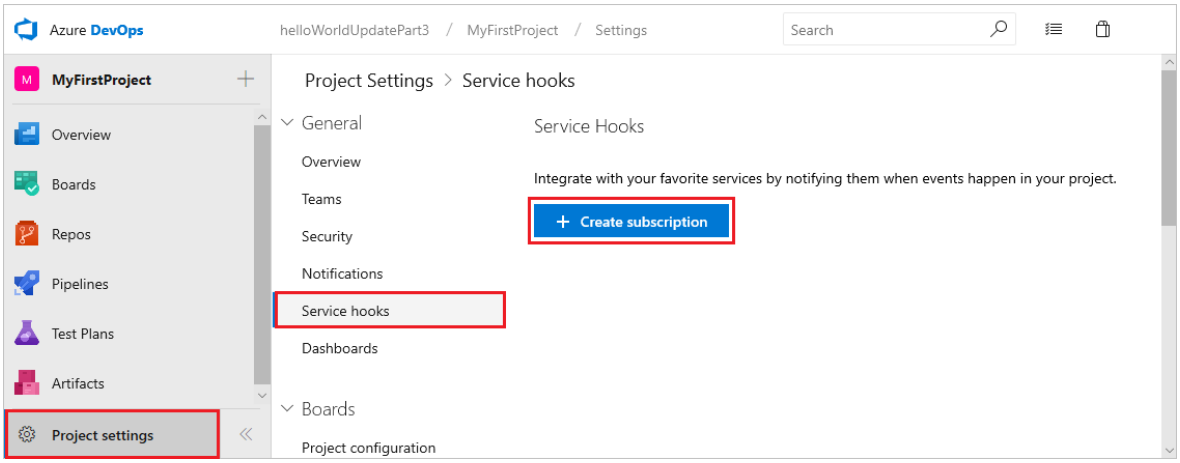
4. Copy the Web Hook URL and go to Azure DevOps.



Create a service hook subscription in your organization

1. Go to your project **Service Hooks** page.

`https://{orgName}/{project_name}/_settings/serviceHooks`

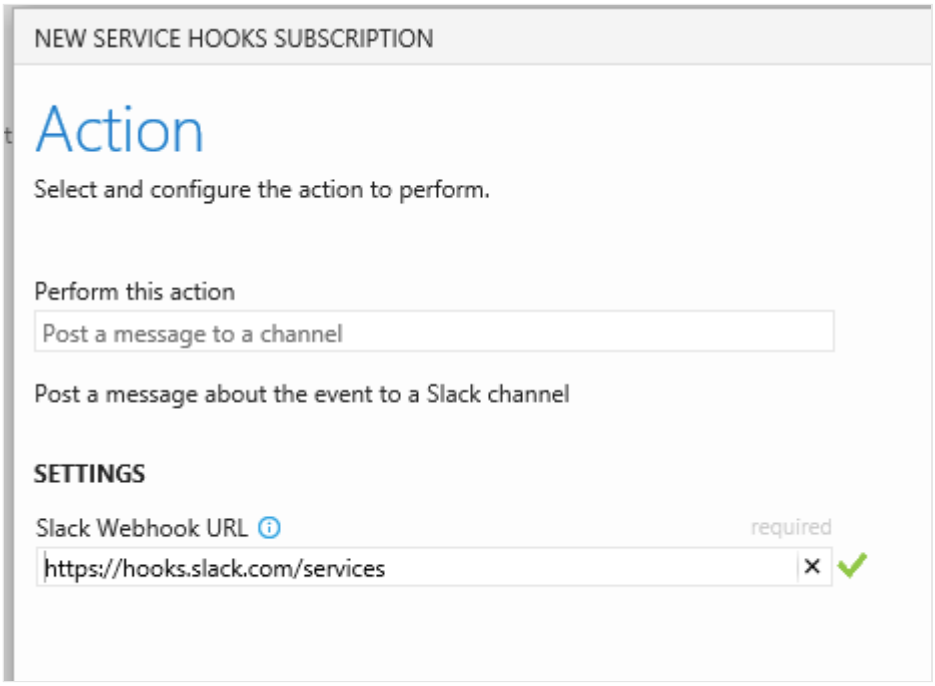


Select **Create Subscription**.

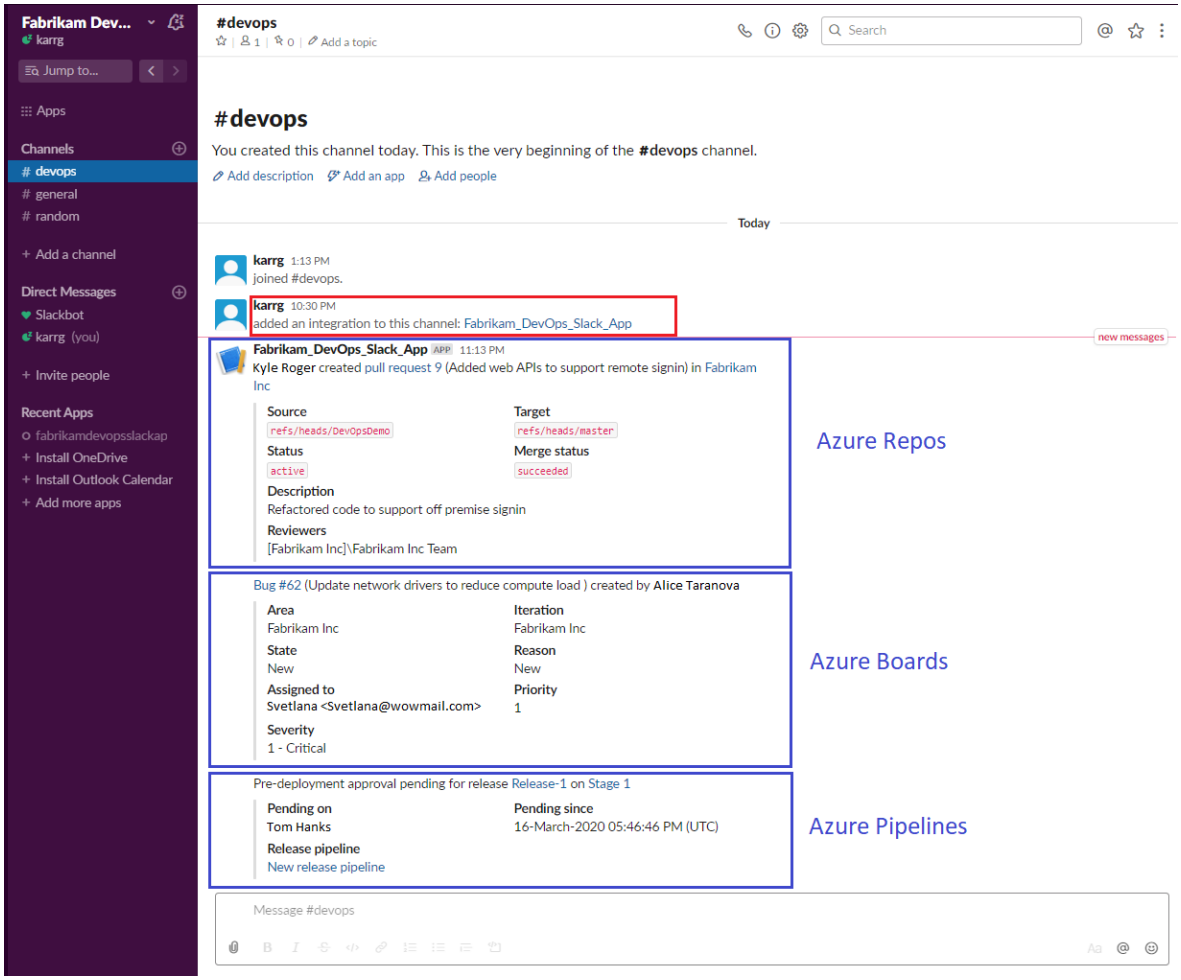
2. Choose the types of events you want to appear in your Slack channel.

   You can filter each of the triggers in specific ways. For example, filter the *pull request created* trigger on the repository in which the pull request occurs, the target branch it applies to, and the team members required or invited to review the request.

3. Paste the Web Hook URL from the Slack integration that you created and select **Finish**.



   When the event occurs in your project, a notification appears in your team's Slack channel.



FAQs

Q: Why don't I have the pull request events as an option when I configure my trigger?

A: Pull requests are only available with projects that use Git. If your project uses TFVC, pull event triggers aren't available, and your code event is called "Code checked in" instead of "Code pushed."

Q: How can I get multiple events to show up in my Slack channel?

A: Create a new subscription for each type of event you want. For example, if you want to see build failures and new work items in your Slack channel, create two more subscriptions.

Feedback

Was this page helpful? Yes No

Provide product feedback

Create a service hook for Jenkins

Article • 08/13/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

This article describes how to set up a service hook for your Azure DevOps project to trigger a Jenkins build. If you use Jenkins to build your apps, you can store your code in a Git repository and use Jenkins for your continuous integration builds. You can trigger a Jenkins build when you push code to your Git repository or when you check in code to a Team Foundation Version Control (TFVC) repository.

You can build part of your app in Azure Pipelines and part in Jenkins. You can trigger a Jenkins build when your Azure Pipelines build is completed so that you use both systems to build your app.

Azure DevOps doesn't charge for setting up service hooks or integrating with external services, and Jenkins is fully open-source and free to use.

Create a Jenkins service hook

To manage service hooks for an Azure DevOps organization or project, you must have **Owner** or **Project Collection Administrator** role in the organization.

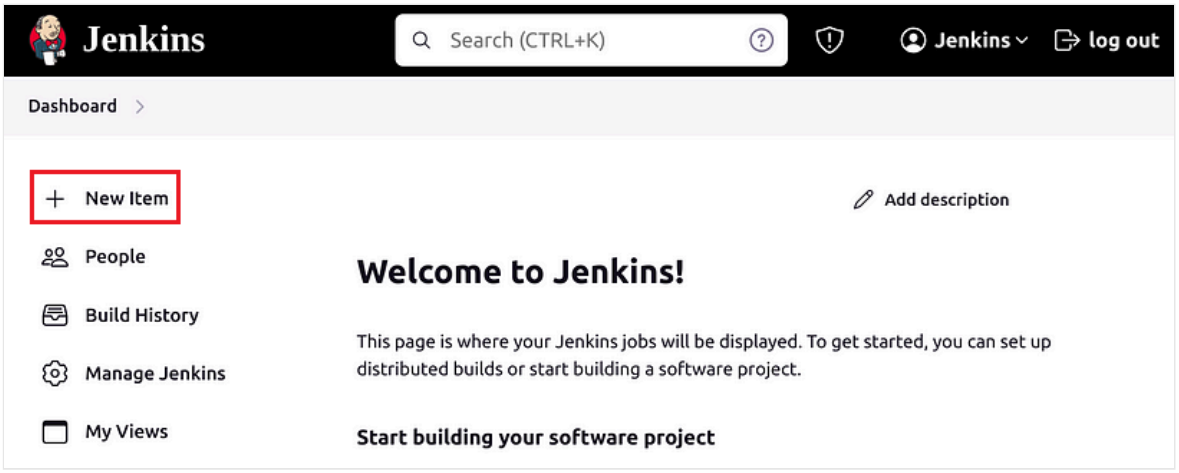
Configure Jenkins

1. If you don't have one, set up a Jenkins server.
2. If you set up Jenkins on-premises, enable HTTPS.
3. In your *jenkins.xml* configuration file, set the hudson.plugins.git.GitStatus.NOTIFY_COMMIT_ACCESS_CONTROL system property to `disabled` by adding or updating the following line in the `<arguments>` tag, before the `-jar` parameter:

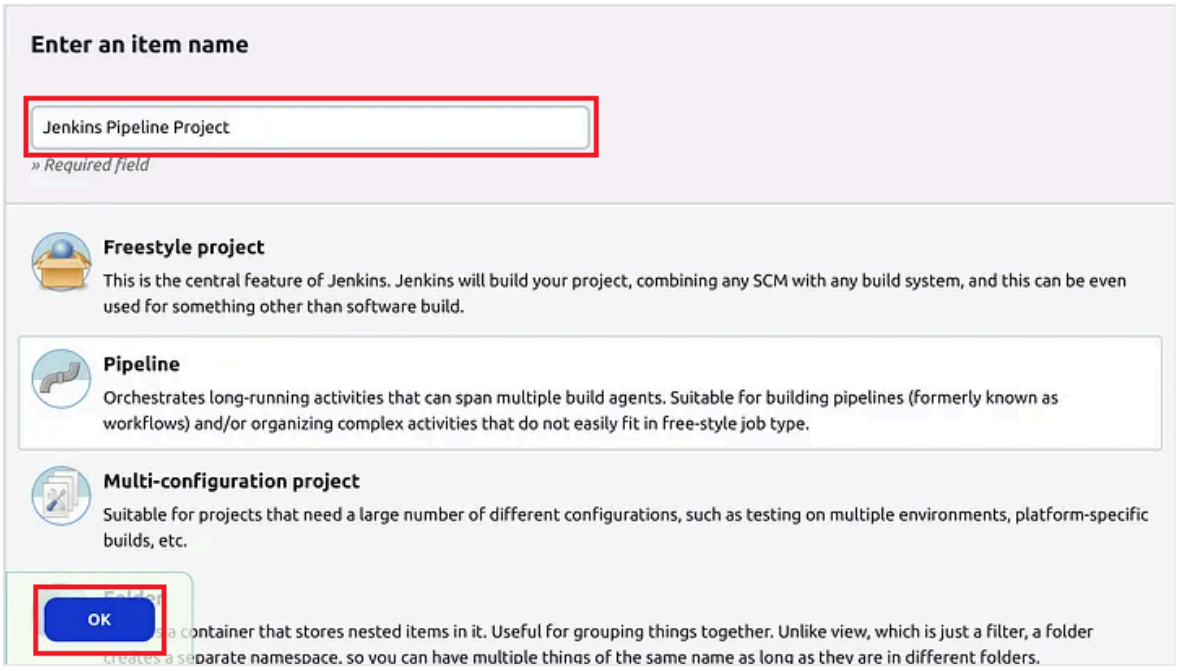
   `-Dhudson.plugins.git.GitStatus.NOTIFY_COMMIT_ACCESS_CONTROL=disabled`.

Set up a Jenkins build

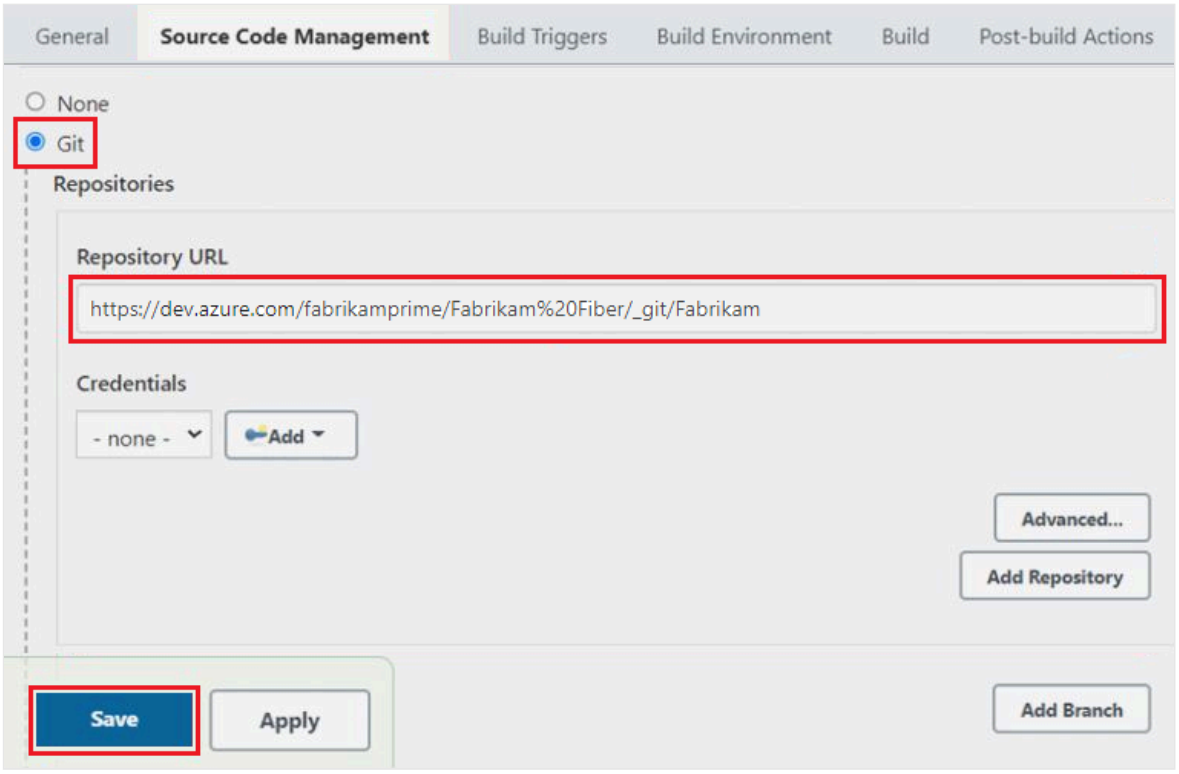
1. In Jenkins, create a new item.



2. Select the type of build that's appropriate for your project.

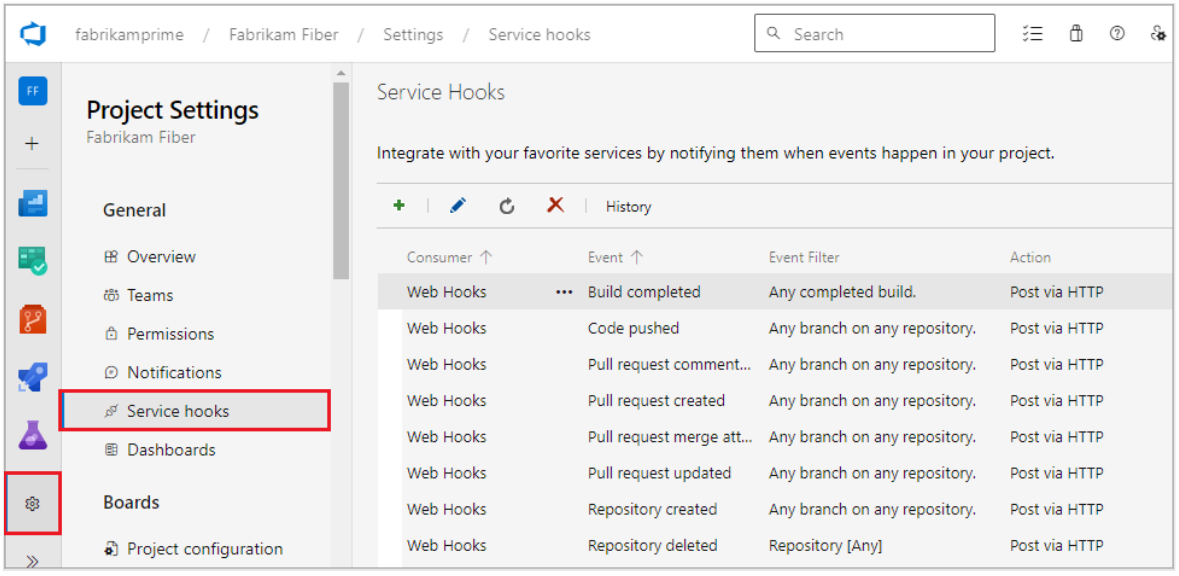


3. Enter the URL for your Git repository.

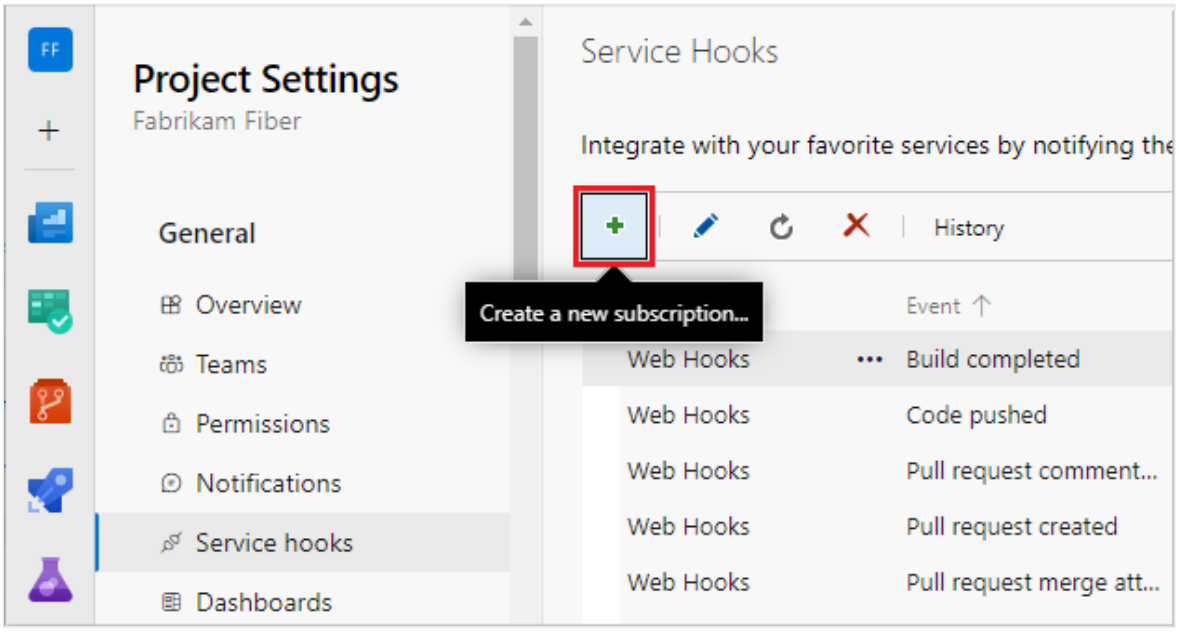


Set up the Jenkins service hook

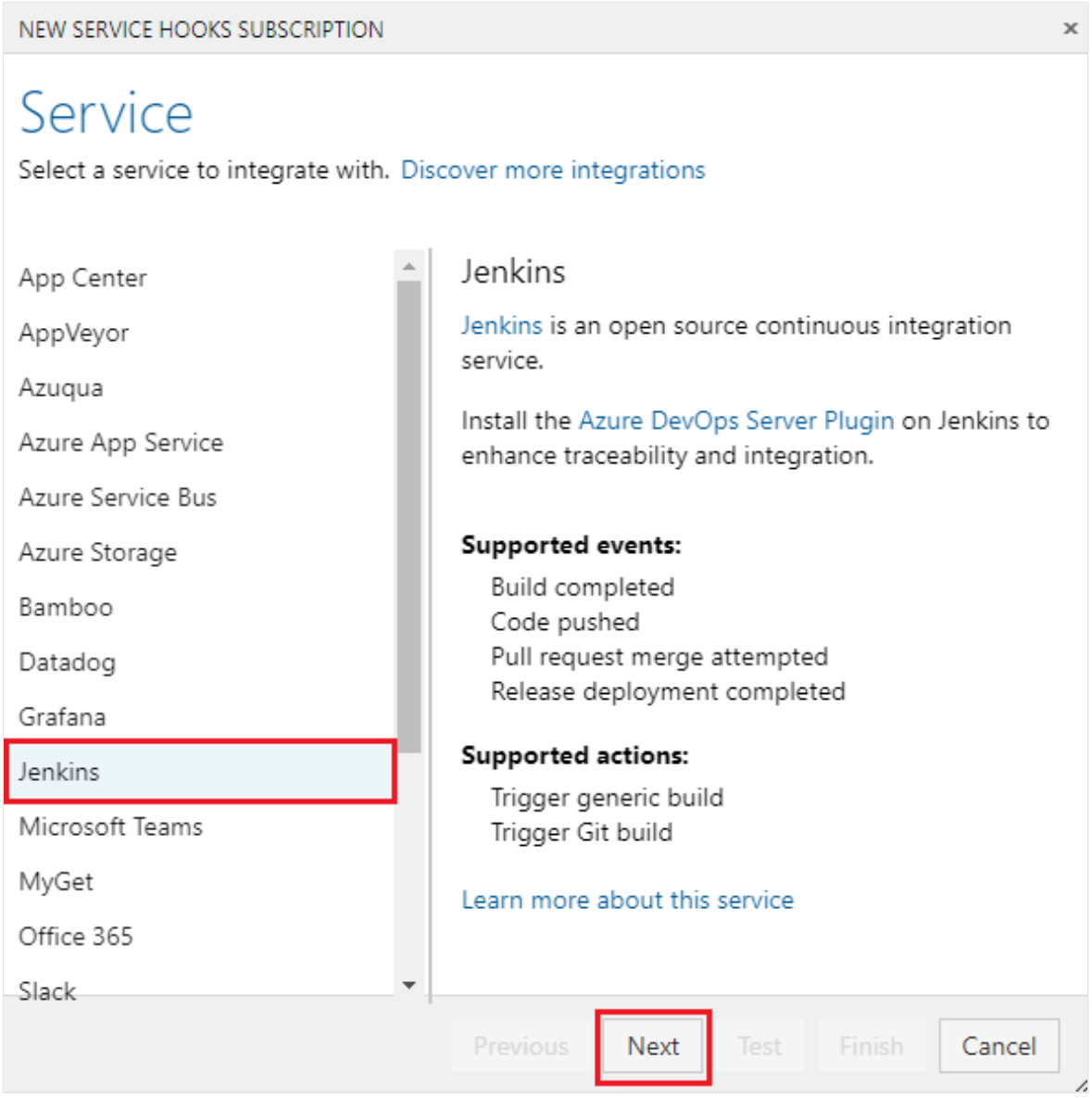
1. In your Azure DevOps project, go to **Project settings** > **Service hooks** at `https://<organization-name>/<project-name>/_settings/serviceHooks`.



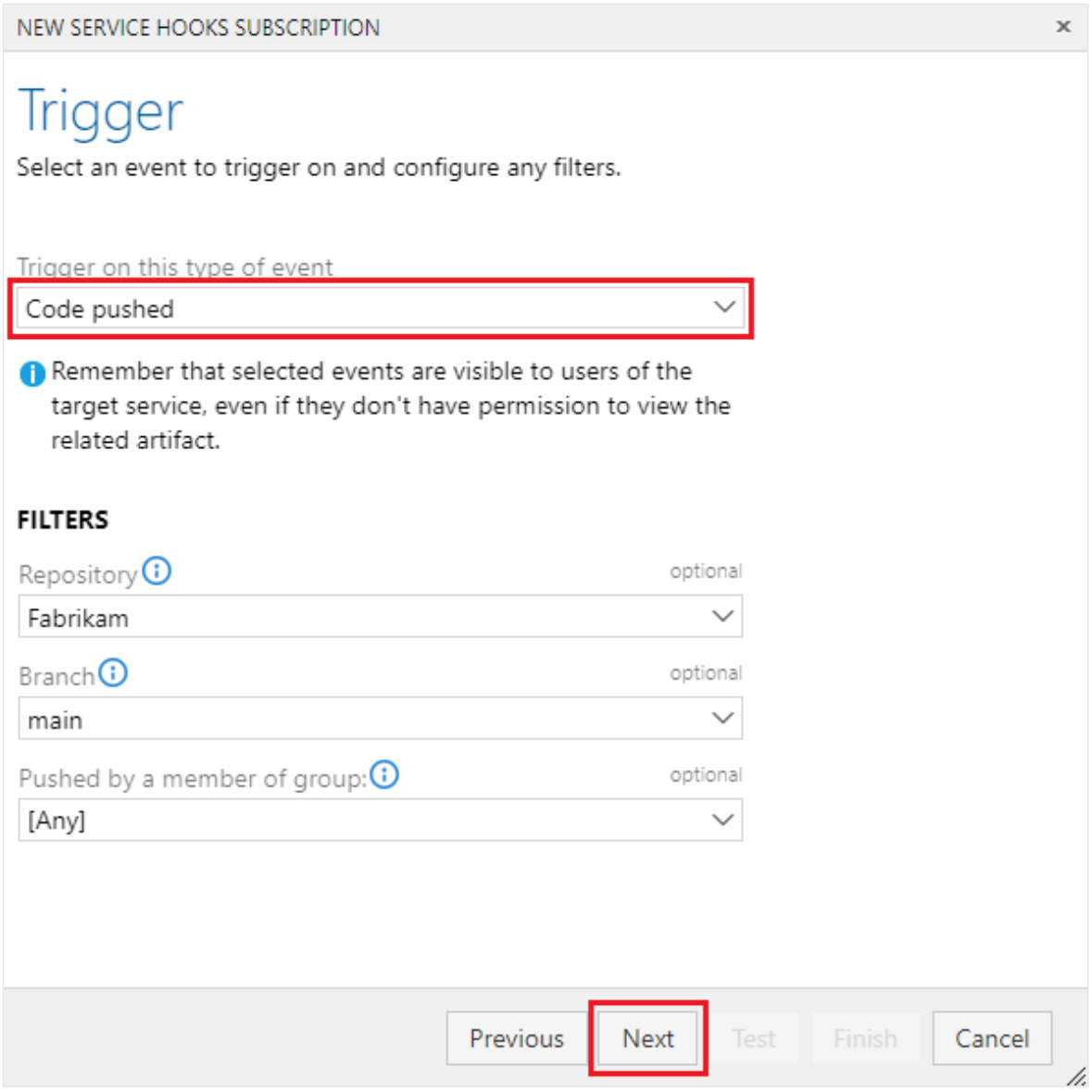
2. On the **Service Hooks** page, select the + icon or **Create subscription**.



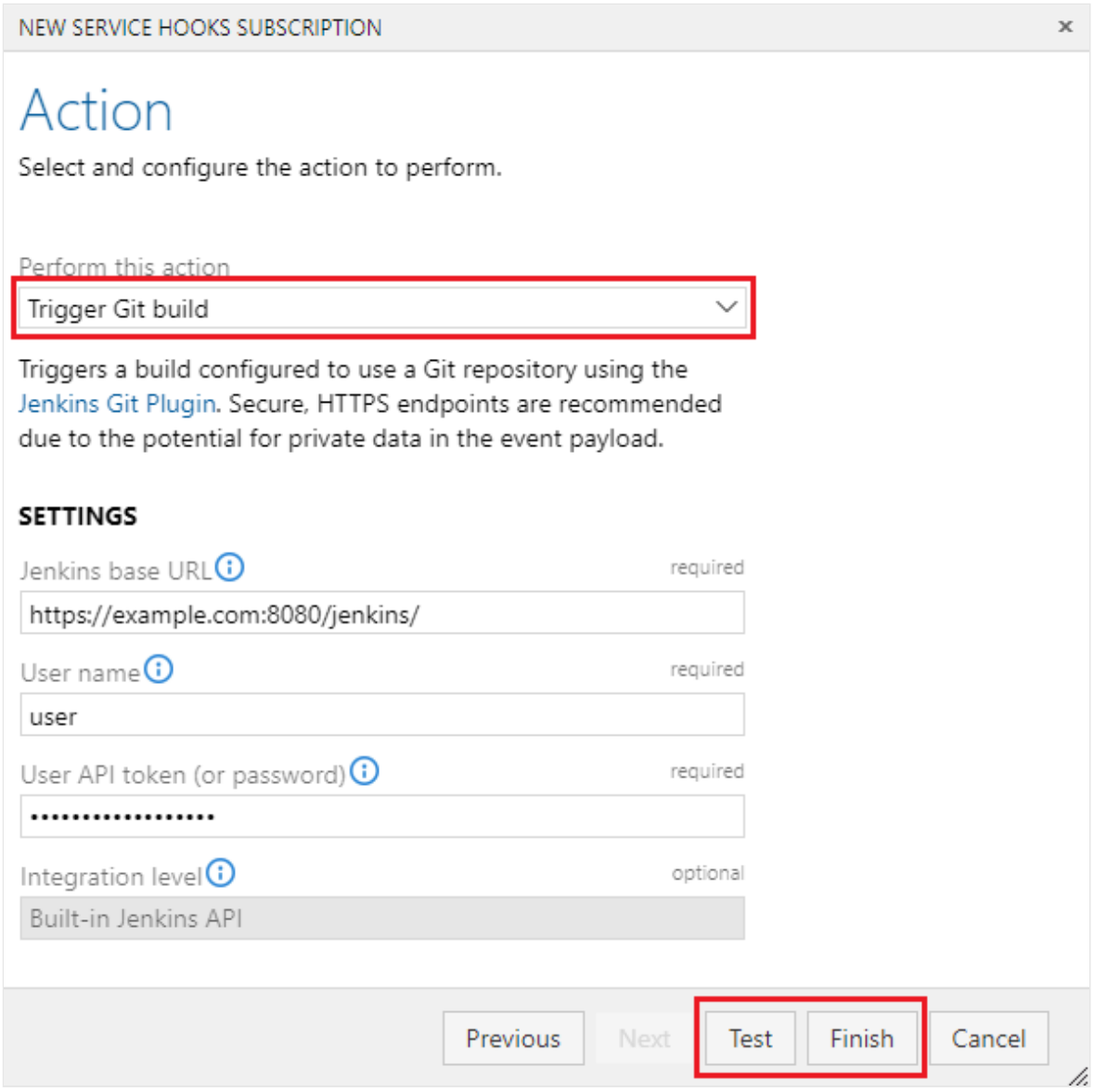
3. On the **Service** screen, select **Jenkins**, and then select **Next**.



4. On the **Trigger** screen, select and configure the Azure DevOps event you want to trigger a Jenkins build, and then select **Next**.



5. On the **Action** screen, configure the Jenkins action to take when the event occurs.



6. Select **Test** to test the service hook, and **Finish** to finish the configuration.

Now when the event occurs in the Git repository, it triggers a Jenkins build.

> **Tip**
>
> You can also create a service hook subscription programmatically with REST APIs. For more information, see Create a service hook subscription programmatically.

Related content

- Integrate with service hooks
- Service hooks events
- Create a service hook subscription programmatically

Feedback

Was this page helpful? Yes No

Provide product feedback

Create a service hook for Azure DevOps Services and TFS with Trello

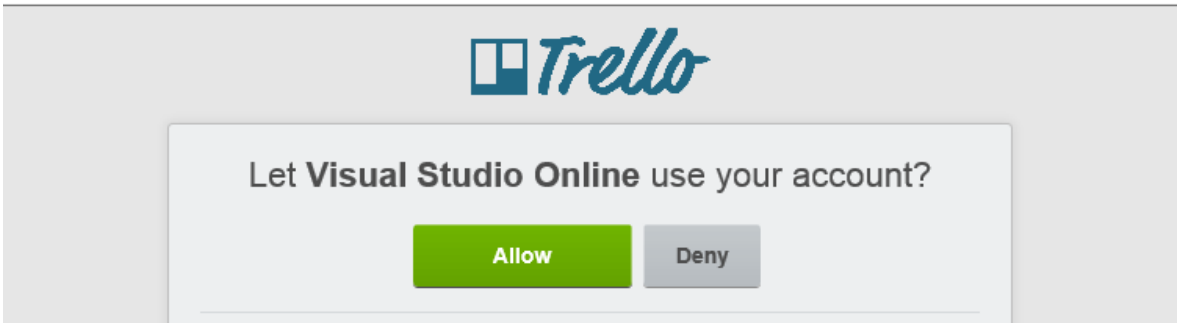
Article • 03/25/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

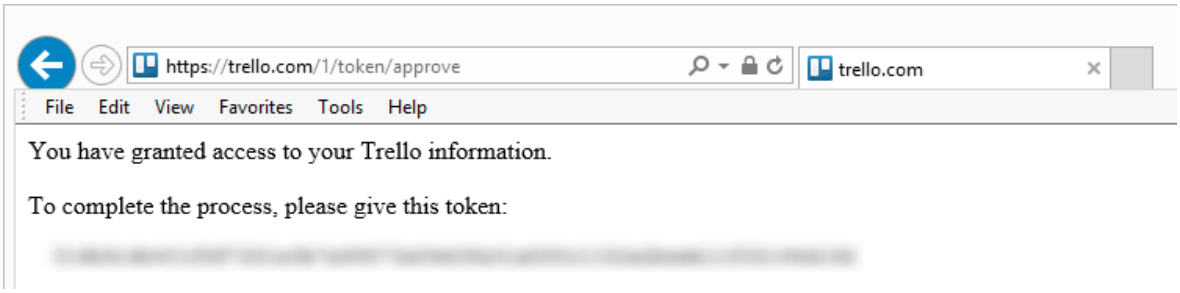
Create cards and lists in Trello in response to events from Azure DevOps. For example, when code is pushed, or a build occurs.

Get a Trello authorization token

1. If you don't have a Trello account, sign up at Trello.
2. Go to the Authorize Azure DevOps Services for Trello page, and sign in with your Trello credentials.
3. Allow Azure DevOps to use your Trello account.



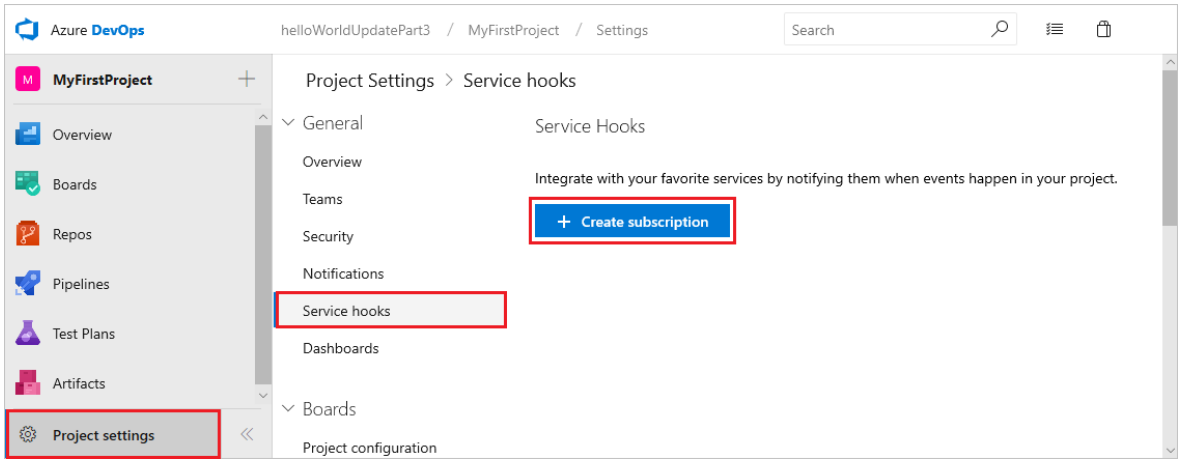
4. Copy the authorization token.



Create a Trello card or list from an Azure DevOps Services event.

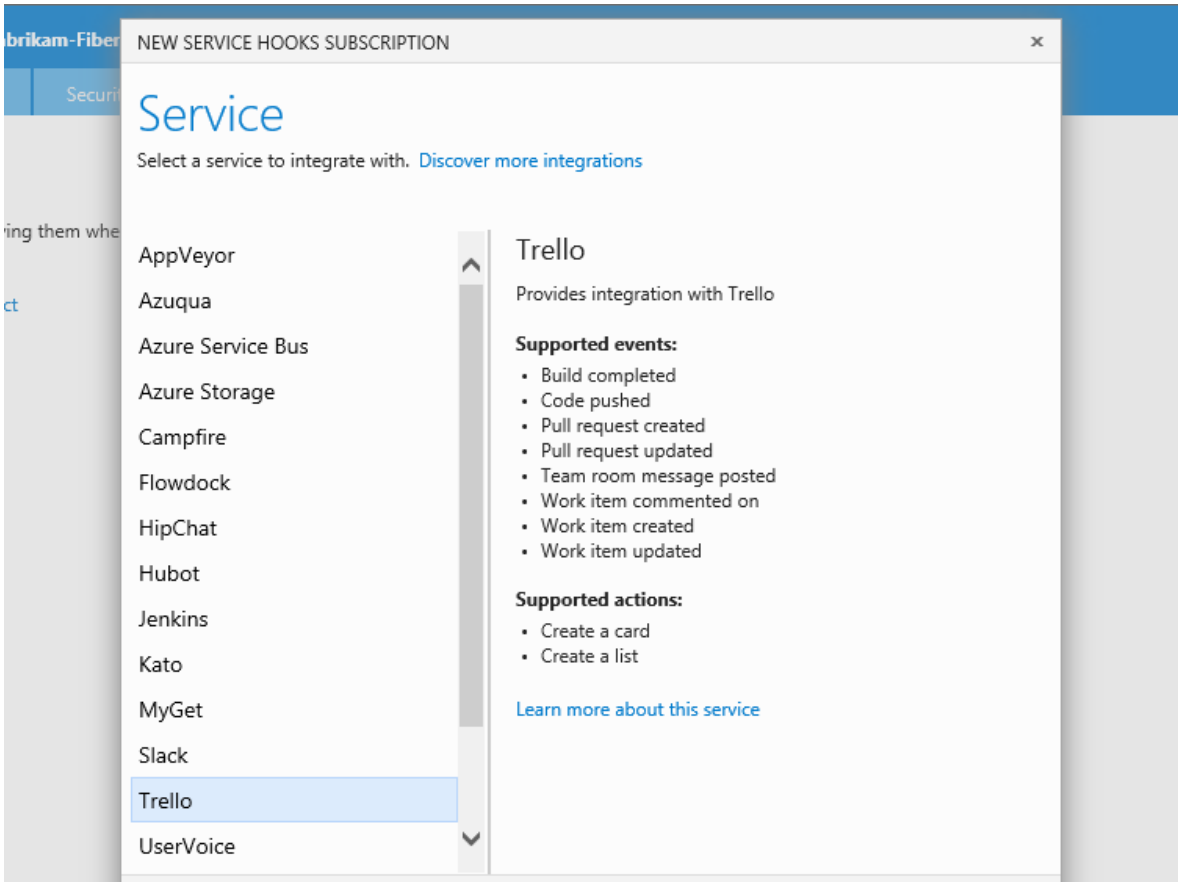
1. Go to your project Service Hooks page:

```
https://{orgName}/{project_name}/_settings/serviceHooks
```

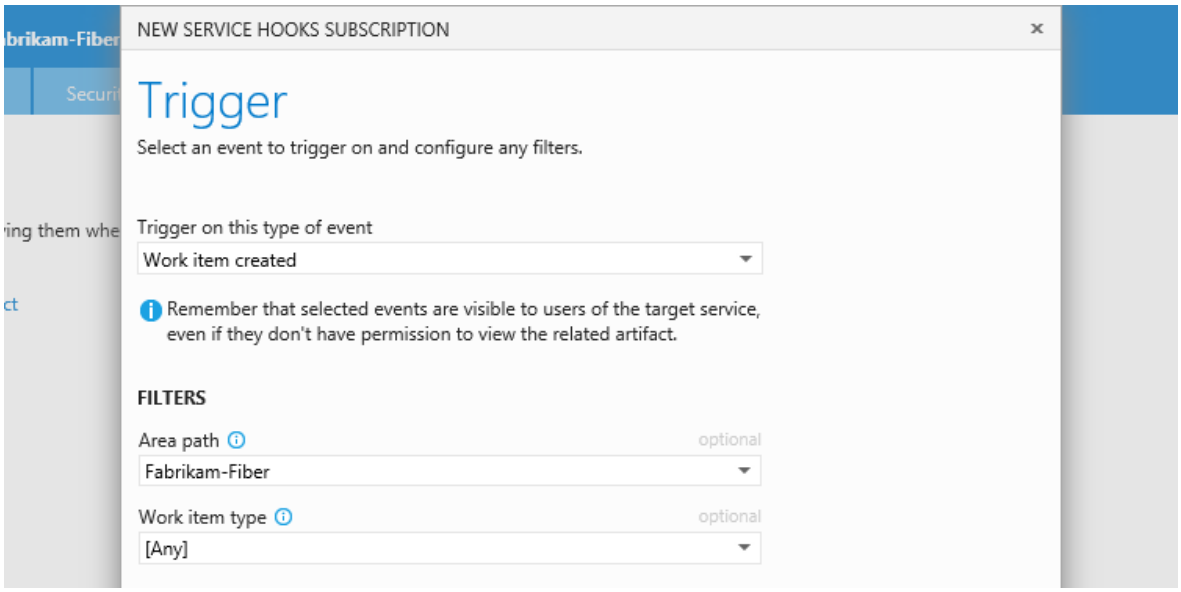


Select **Create Subscription**.

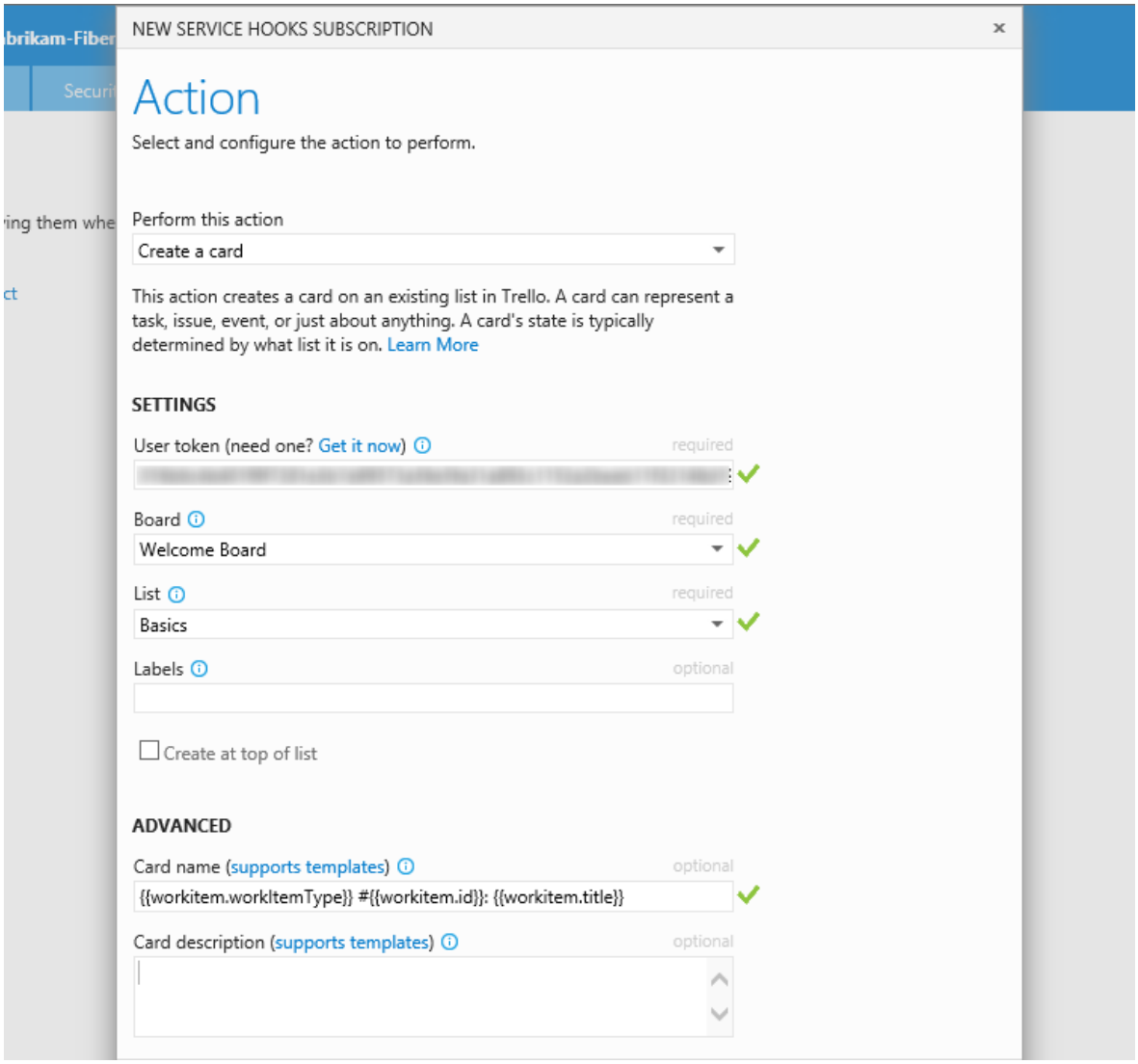
2. Pick the Trello service.



3. Configure the triggering Visual Studio event. In this case, we're going to have the subscription respond when a work item is created.



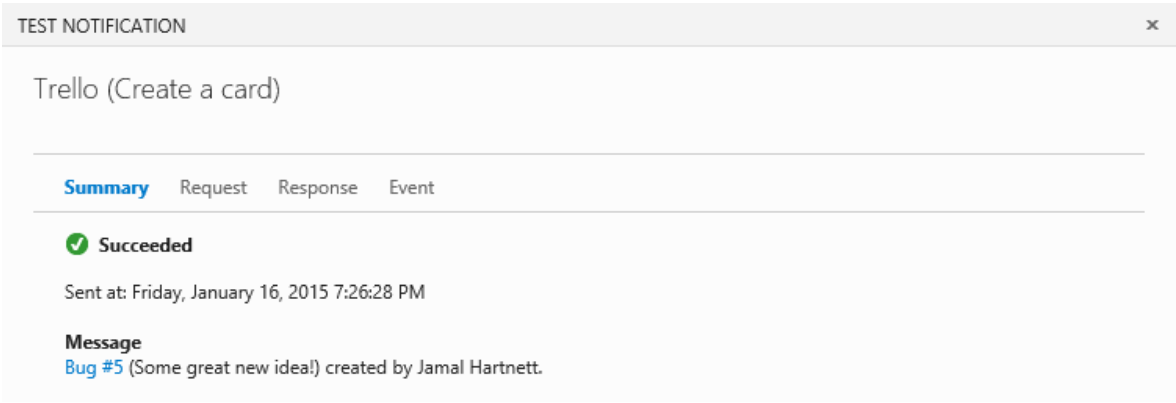
4. Set up the action that Trello will take in response to the trigger - either create a card or a list.



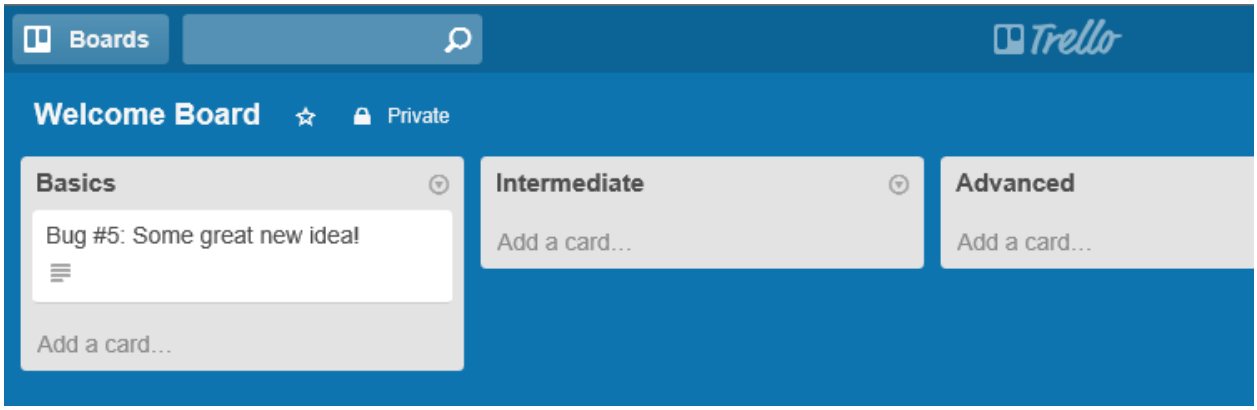
You can use placeholders to insert content from the event into the name and description of the cards or lists that the subscription creates. For example, when we run the test notification, the card that gets created is named "Bug #5: Some great

new idea!" because the test work item is a bug (ID=5) with the title "Some great new idea!".

5. Test the service hook subscription and finish the wizard.



Now it's set up. Go to Trello and see the cards appear.



Placeholders

You can insert placeholders in the name of a list or the name or description of a card to be created by your subscription. When a list or card is created, the placeholders will be replaced by values from the event that was raised. The example we used for the card name in this page uses:

```
{{workitem.workItemType}} #{{workitem.id}}: {{workitem.title}}
```

So when a bug is created with the ID 5 and title "Some great new idea!", the card name is "Bug #5: Some great new idea!".

The basic form of the placeholder is `{{resource.field}}` where resource is the name of the resource raising the event (work item, build, and so on) and field is a field within the

resource section of the event, like ID. So, if the subscription is for a completed build, it might be something like

```
    Build {{build.id}} completed at {{build.finishTime}}
```

Where build is the name of the resource and ID, finishTime are the fields available in this resource. To understand the event types, fields and resources available to use, look at the events reference.

Work item fields

Work item fields appear in the event in the fields array, like this example:

```
    " fields": {
                " System.AreaPath": "Fabrikam-Fiber-Git",
                " System.TeamProject": "Fabrikam-Fiber-Git",
                " System.IterationPath": "Fabrikam-Fiber-Git",
                " System.WorkItemType": "Product Backlog Item",
                " System.State": "New",
                " System.Reason": "New backlog item",
                " System.CreatedDate": "2014-11-05T21:11:28.85Z",
                " System.CreatedBy": "Normal Paulk",
                " System.ChangedDate": "2014-11-05T21:11:29.23Z",
                " System.ChangedBy": "Normal Paulk",
                " System.Title": "Test PBI",
                " Microsoft.Azure DevOps Services.Common.BacklogPriority":
999999999,
                " WEF_6CB513B6E70E43499D9FC94E5BBFB784_Kanban.Column": "New"
            },
```

Working directly from the event definition, we would have created our card name like this example:

```
    {{workitem.fields["System.workItemType"]}} #
{{workitem.fields["System.id"]}: {{workitem.fields["System.title"]}}
```

As a shortcut, you can reference any fields in the System or Microsoft.Azure DevOps Services.Common namespaces as if they were fields of the resource. So `{{workitem.fields["System.workItemType"]}}` becomes `{{workitem.workItemType}}`.

Placeholder expressions

These placeholders use a handlebars templates implementation that is largely compatible with Mustache. Some useful expressions include

Expand table

Type of expression	examples
basic expressions	`{{workitem.name}}`
array expressions	`{{pullRequest.reviewers.[0].displayName}}`
Mustache sections	`{{#workitem.assignedTo}}` This WI is assigned `{{/workitem.assignedTo}}`
Mustache Inverted Sections	`{{^workitem.assignedTo}}` This WI isn't assigned `{{/workitem.assignedTo}}`
Handlebars block helpers	with if/else unless each
Handlebars paths	... this For example, `{{../comment/id}}` or `{{this/title}}`
Template comments	`{{!-- this is a handlebar comment --}}`

Pricing

Azure DevOps doesn't charge for the framework for integrating with external services. Check out the specific service's site for pricing related to their services.

Q & A

Q: Can I programmatically create subscriptions?

A: Yes, see details here.

Q: Can I get more information about Trello?

A: Yes, trello.com.

Feedback

Was this page helpful? Yes No

Provide product feedback

Create a service hook for Azure DevOps with Datadog

Article • 01/16/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Learn how to create events and metrics in Datadog in response to events from Azure DevOps. Use these metrics and events in Datadog to create dashboards, troubleshoot issues, and create monitors to alert you of critical issues. Datadog accepts all Azure DevOps event types.

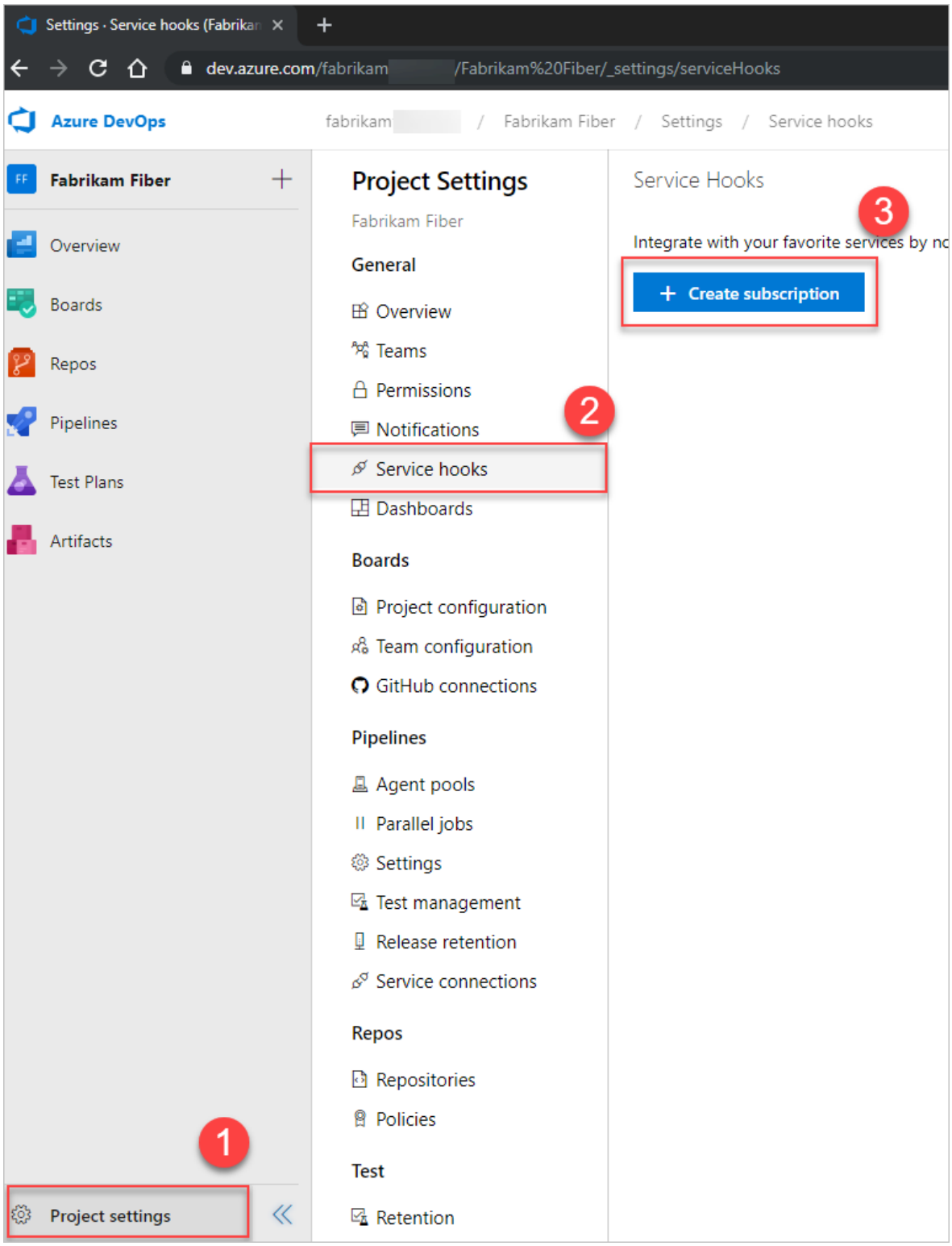
> **ⓘ Important**
>
> The Datadog feature may not be turned on by default for Azure DevOps Server 2019 and 2020. This is a known issue. Until it's resolved, you can use the following SQL command in your config database to turn on the feature: `exec prc_SetRegistryValue 1, '#\FeatureAvailability\Entries\ServiceHooks.Consumers.datadog\AvailabilityState\', 1`

Prerequisites

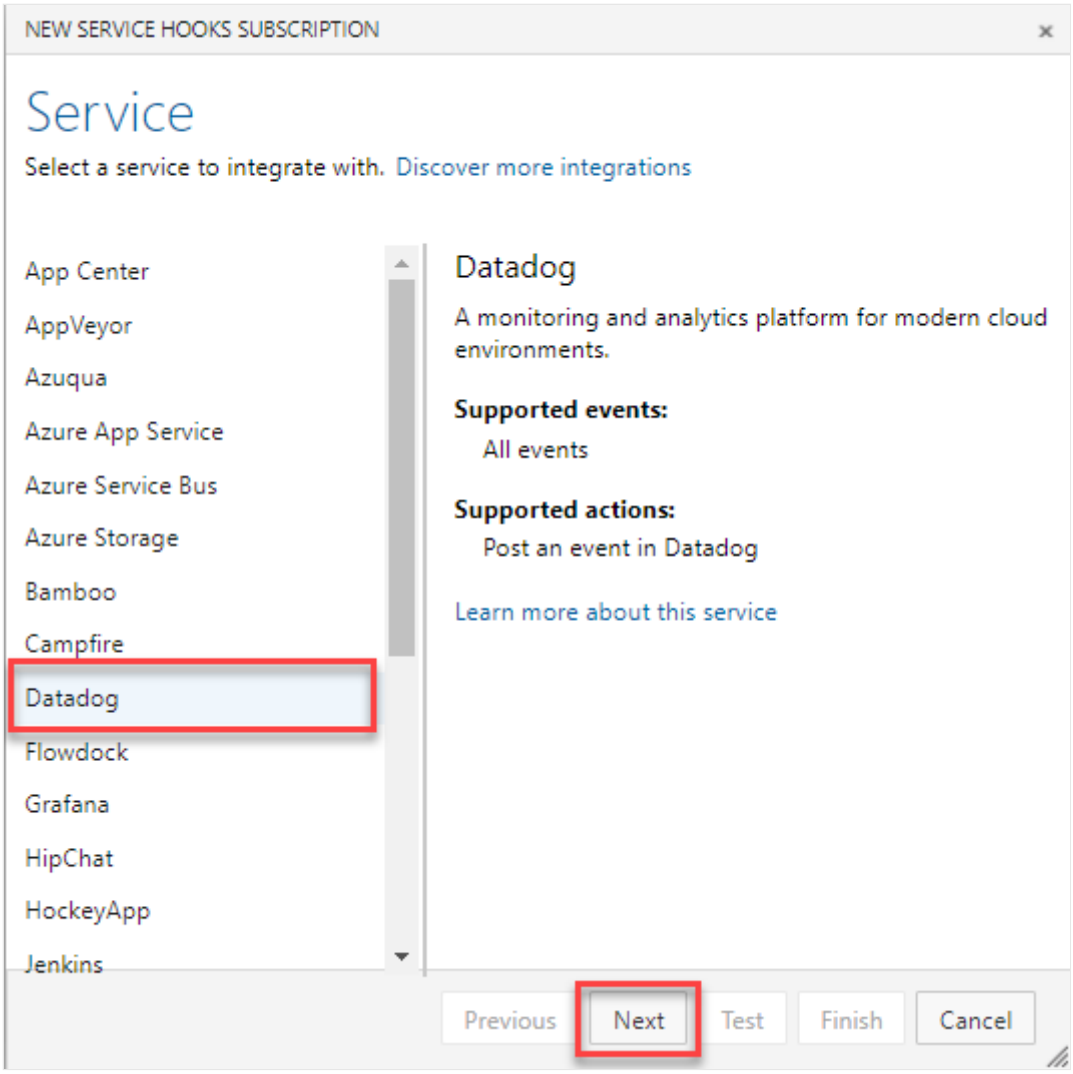
1. You must have *Edit subscriptions* and *View subscriptions* permissions. By default, only project administrators have these permissions. To grant the permissions to other users, you can use the command-line tool or the Security REST API.
2. If you don't have a Datadog account, get started with Datadog. a. In the Datadog application, go to Integrations > API. b. Create a new key or select an existing one, and then copy the key to your clipboard.

Start sending Azure DevOps events to Datadog

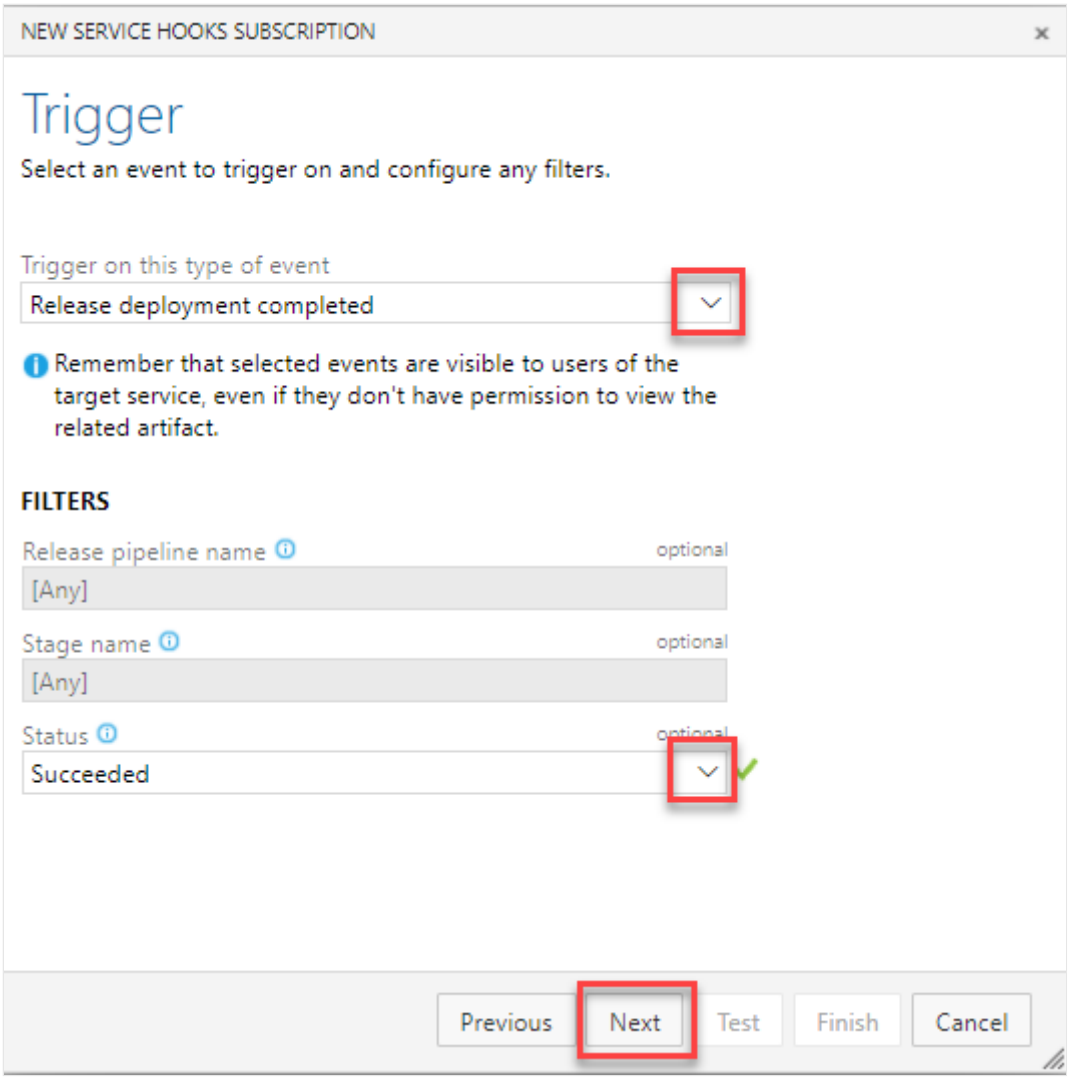
1. Go to **Project settings** > **Service hooks**: `https://{orgName}/{project_name}/_settings/serviceHooks`.
2. Select **Create subscription**.



3. Choose **Datadog** among the list of services, and then choose **Next**.



4. Select an event to trigger on, configure any filters, and then select **Next**.

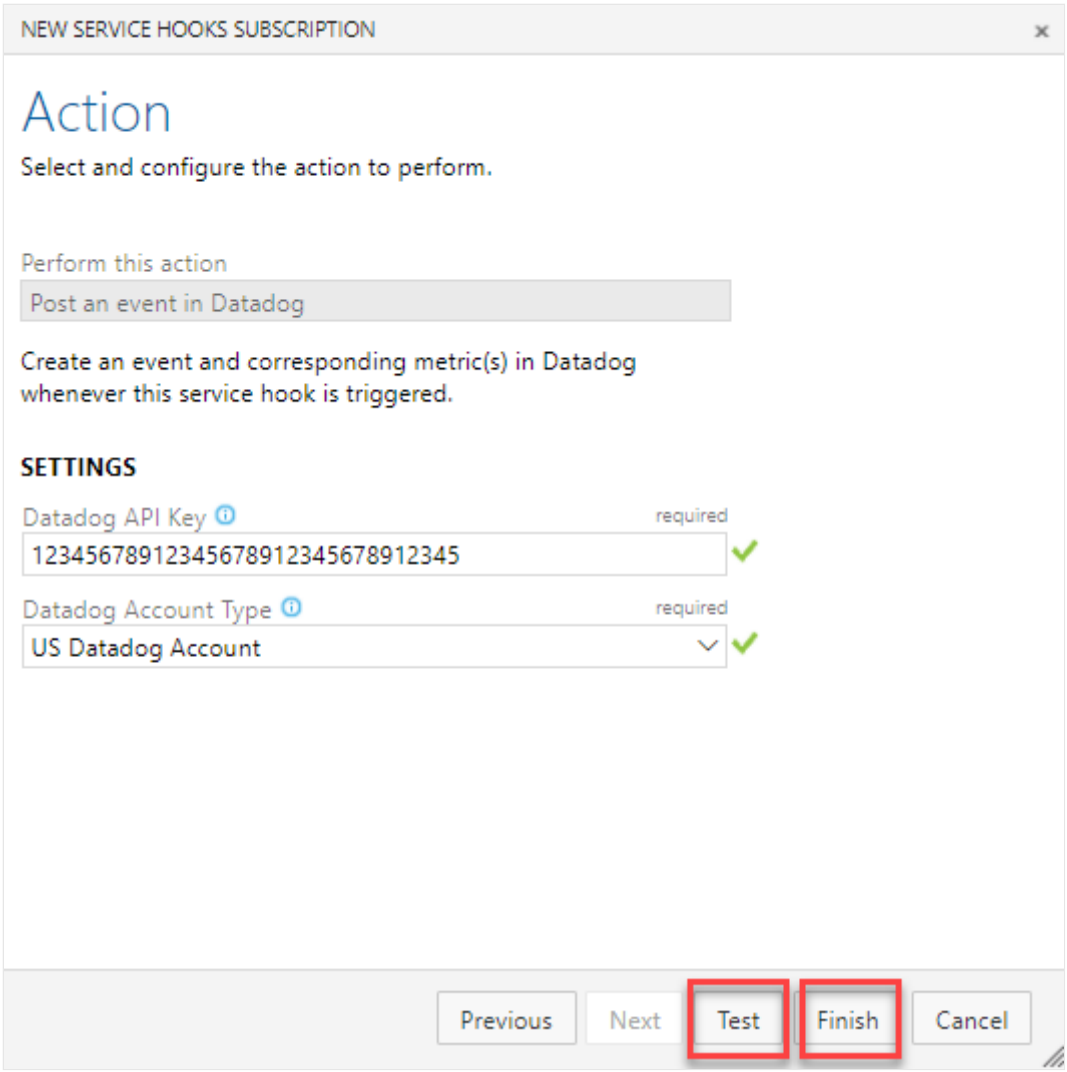


5. Configure the action to perform.

    - Paste (ctrl+v) your Datadog API Key into the required field.
    - Indicate whether your Datadog account is US or EU.

6. Choose **Test** to verify that Azure DevOps can use the provided configuration and successfully create a subscription.

7. Once verified, choose **Finish** to complete the creation of subscription.



8. Repeat steps 2-7 for each event type you want to send to Datadog. Datadog accepts and encourages users to send all event types.

9. Go to Datadog to see events and metrics start to flow into your environment.

FAQs

Q: Can I create service hook subscriptions programmatically?

A: Yes. For more information, see Create a service hooks subscription programmatically. Submit to either of the following Datadog endpoints:

- **US:** `https://app.datadoghq.com/intake/webhook/azuredevops?api_key=<API_KEY>`
- **EU:** `https://app.datadoghq.eu/intake/webhook/azuredevops?api_key=<API_KEY>`
- **US3:** `https://us3.datadoghq.com/intake/webhook/azuredevops?api_key=<API_KEY>`
- **US5:** `https://us5.datadoghq.com/intake/webhook/azuredevops?api_key=<API_KEY>`
- **AP1:** `https://ap1.datadoghq.com/intake/webhook/azuredevops?api_key=<API_KEY>`

- **Gov:** `https://app.ddog-gov.com/intake/webhook/azuredevops?api_key=<API_KEY>`

Q: How can I use these events in Datadog?

A: Azure DevOps events that are sent to Datadog are useful for creating dashboards, setting up monitors, and finding correlations during troubleshooting. You can also use them to get insights into how your developer operations processes impact application performance.

Q: What event types can I send to Datadog?

A: Datadog accepts all event types.

Q: Can I get more general information about Datadog?

A: Yes, see datadoghq.com .

Azure DevOps integration with Microsoft Teams

Article • 08/13/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

This article describes how you can integrate your Azure DevOps project activities directly into your Microsoft Teams channels. For example, you can see the following Azure DevOps events in your Teams channels:

- Work item updates
- Pull requests
- Code commits
- Builds
- Release deployments and approvals

Requirements and limitations

- To create Azure DevOps subscriptions in Teams, you must have **Project Administrator** permissions in your Azure DevOps project.
- To receive Azure DevOps notifications in Teams, you must have **Third party application access via OAuth** enabled in your Azure DevOps organization settings.
- Only Azure DevOps organizations in the same Microsoft Entra tenant as your Microsoft Teams account can be connected. Even if your email addresses are the same for Azure DevOps and Microsoft Teams, they can't be linked if they're backed by different tenants. For more information, see Access via Microsoft Entra FAQs.
- Office 365 Connectors within Teams are being retired, and you might be unable to create new connectors. For more information, see Retirement of Office 365 connectors within Microsoft Teams. Features that provide similar functionality to Office 365 Connectors with better scalability and security include Workflows or the Azure DevOps Services apps for Teams.

Azure DevOps Services apps for Teams

> **Note**

This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. For more information, see Azure DevOps Feature Timeline.

You can use the following apps that offer rich features to integrate with Microsoft Teams.

Azure Boards app for Teams

The Azure Boards app for Teams helps you easily create and monitor work items from your Teams channels. You can create work items by using a command, or use message actions to convert conversations in the channel into work items. You can also set up and manage subscriptions to get notifications in your channel whenever work items are created or updated. For more information, see Use the Azure Boards app in Microsoft Teams.

Azure Pipelines app for Teams

The Azure Pipelines app for Teams helps you easily monitor events in your pipelines from your Teams channels. You can set up and manage subscriptions for completed builds, releases, and pending approvals, and get notifications for these events in your channels. You can also approve builds and release deployments from your channels. For more information, see Integrate Azure Pipelines with Microsoft Teams.

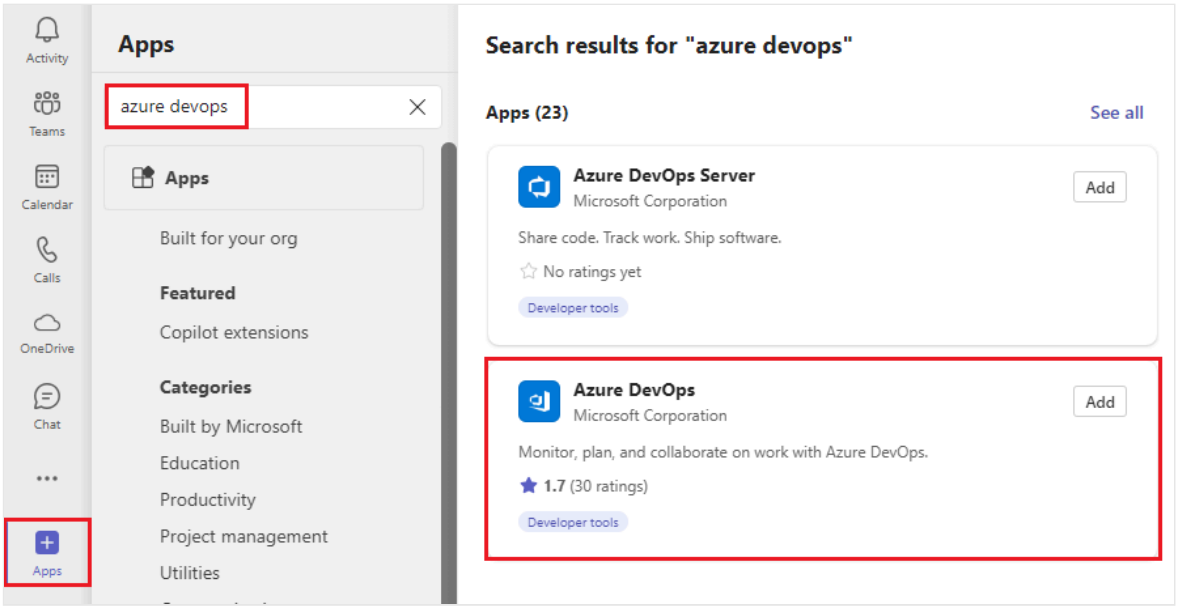
Azure Repos app for Teams

The Azure Repos app for Teams helps you easily monitor events in your repositories from your Teams channels. You can set up and manage subscriptions for code commits, pull request (PR) creation, and PR updates, and get notifications for these events in your channels. For more information, see Use Azure Repos with Microsoft Teams.

Add and configure the Azure DevOps tab in Teams

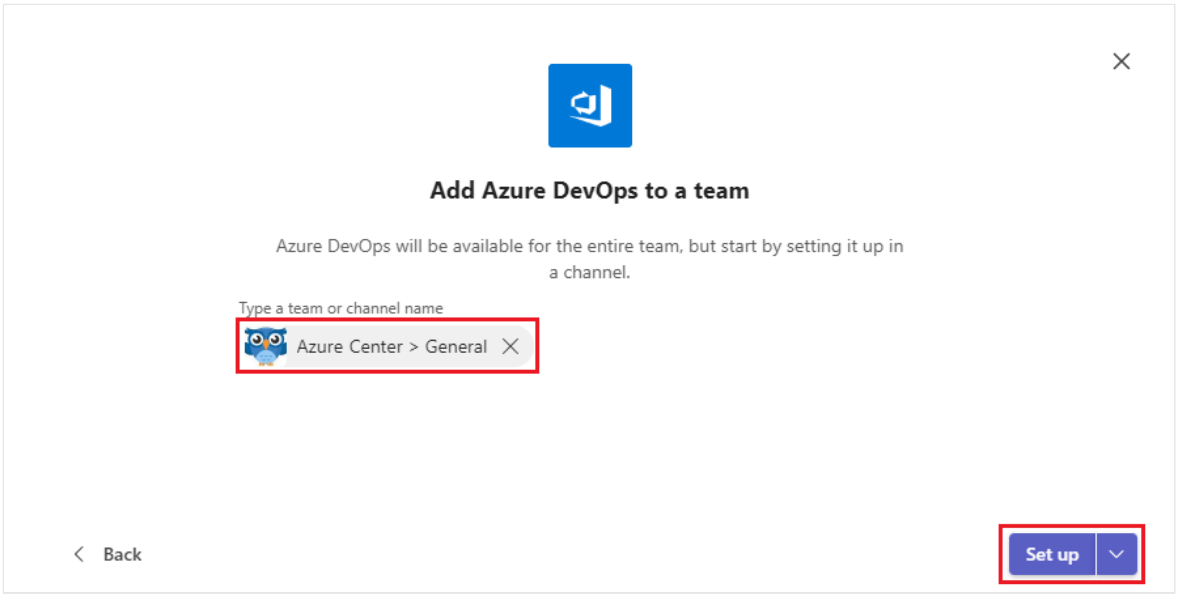
To bring your project dashboard or Kanban board into Teams, you can install the Azure DevOps app in a tab in your Teams channel. The Azure DevOps app lets you insert content from the app in messages, and get notifications from the app in your channels.

1. In Teams, select **Apps** from the left menu and then search for *Azure DevOps*.

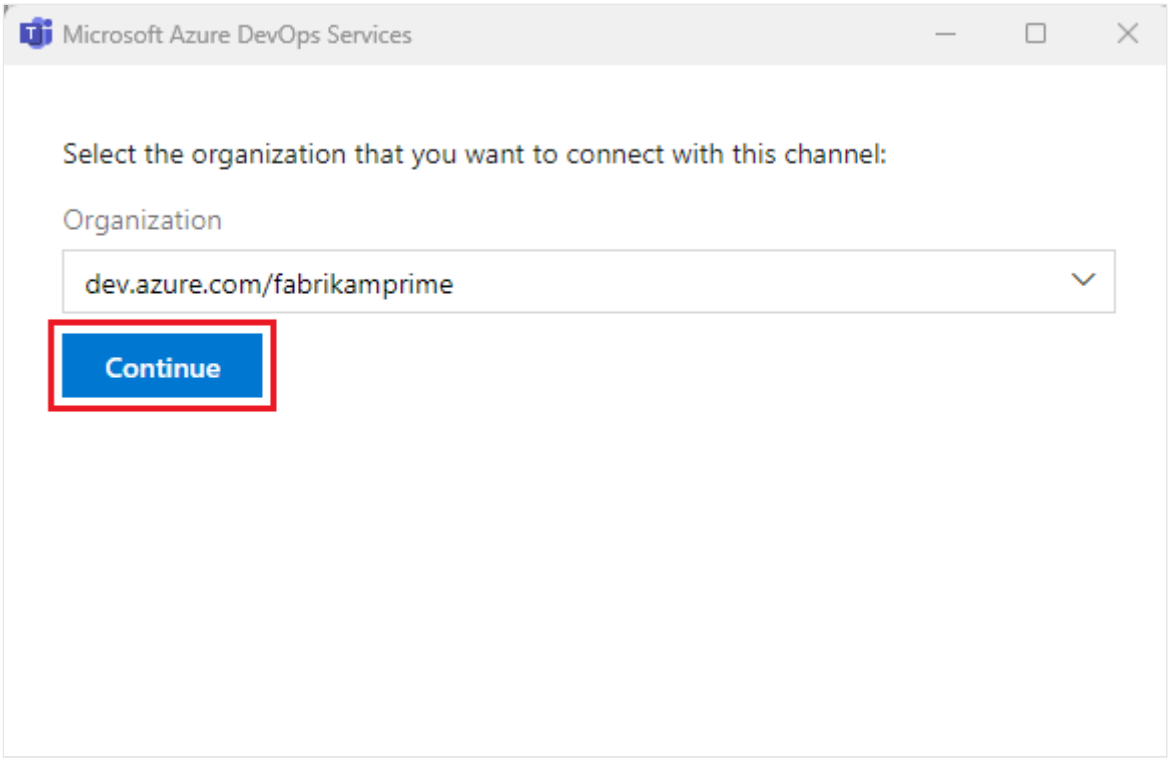


2. Select **Azure DevOps**, and then select **Add to a team**.

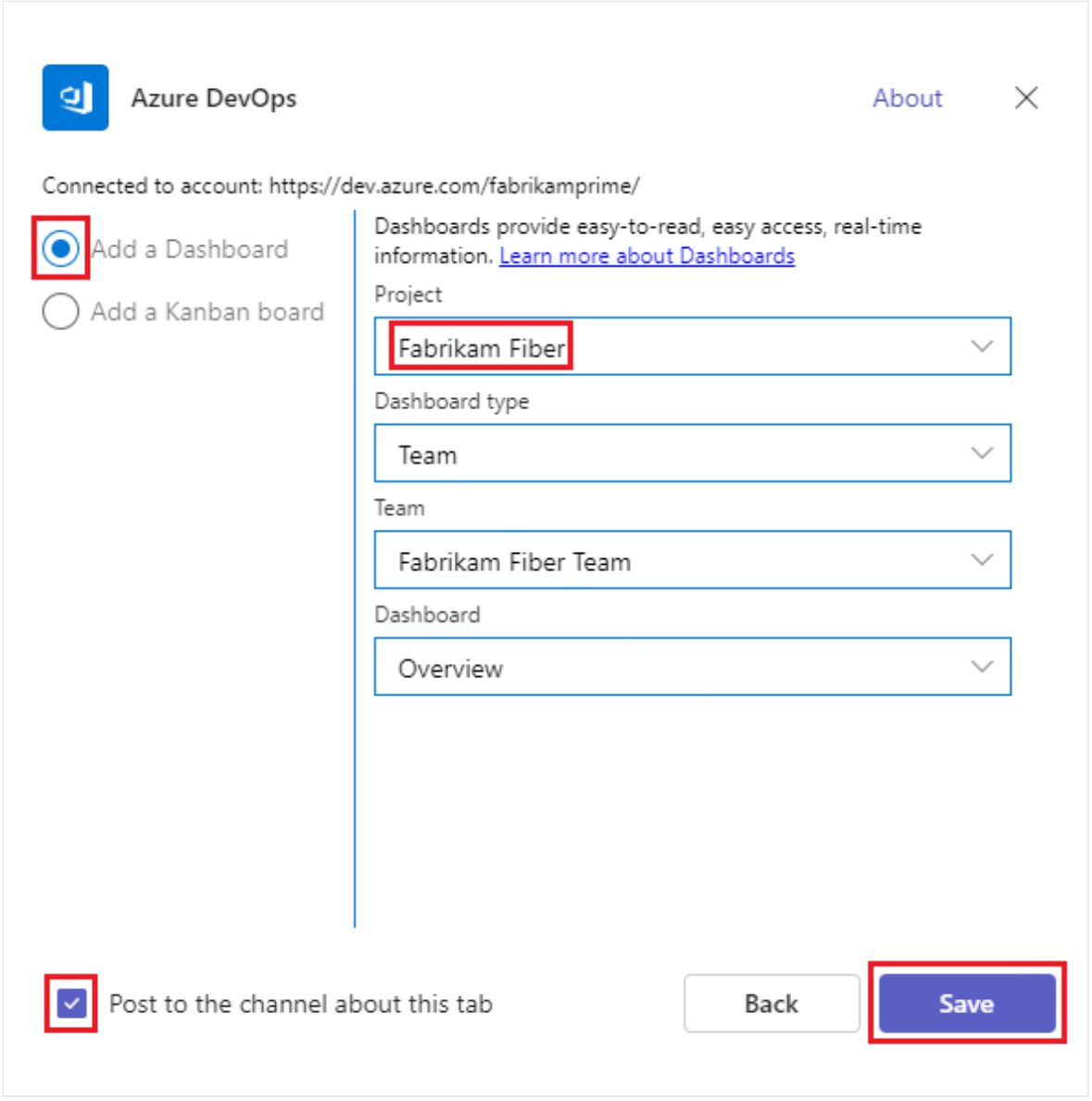
3. Select the Teams channel to add to the app to, and then select **Set up**.



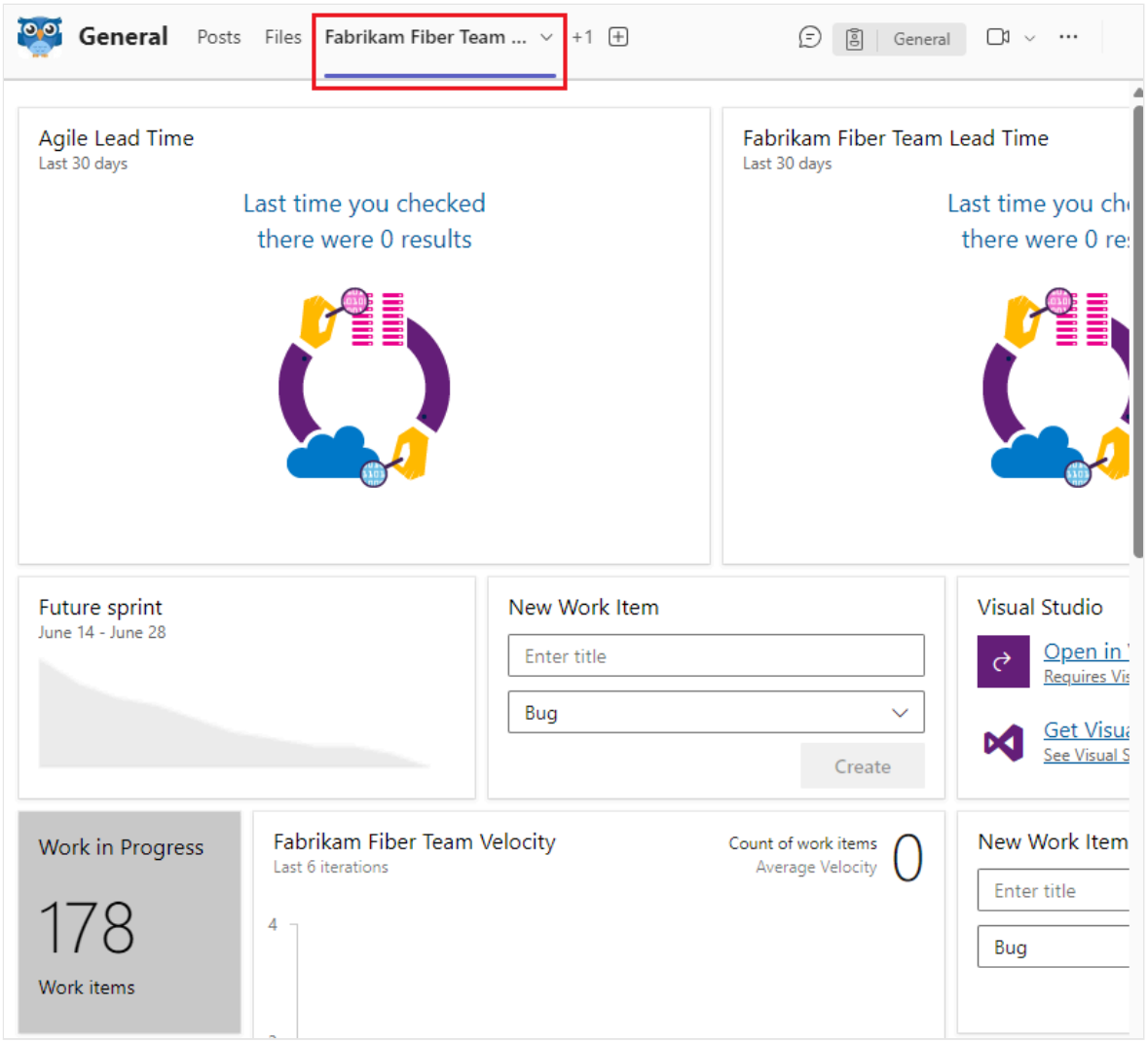
4. Select **Select organization**, select your organization, and then select **Continue**.



5. On the **Azure DevOps** screen, select a **Project**, and whether to add a **Dashboard** or a **Kanban board** to the tab. Select other configurations depending on your choice, and select whether you want to post to the channel about adding the tab.



6. Select **Save**. The new tab and board appear in your channel.



To manage your Teams apps and tabs, select **Apps** in the Teams menu and then select **Manage your apps** at the bottom of the **Apps** panel.

Related content

- Use the Azure Boards app in Microsoft Teams
- Integrate Azure Pipelines with Microsoft Teams
- Use Azure Repos with Microsoft Teams
- Install the Workflows app in Microsoft Teams

Feedback

Was this page helpful? Yes No

Provide product feedback

Webhooks

Article • 12/19/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

This article describes webhooks and how to set them up for your Azure DevOps project. Webhooks provide a way to send a JSON representation of an Azure DevOps event to any service that has a public endpoint.

> **Note**
>
> Azure DevOps doesn't charge for setting up service hooks or integrating with external services. Refer to the specific service's site for pricing related to their services.

Prerequisites

You need the following prerequisites to manage webhooks for an Azure DevOps organization:

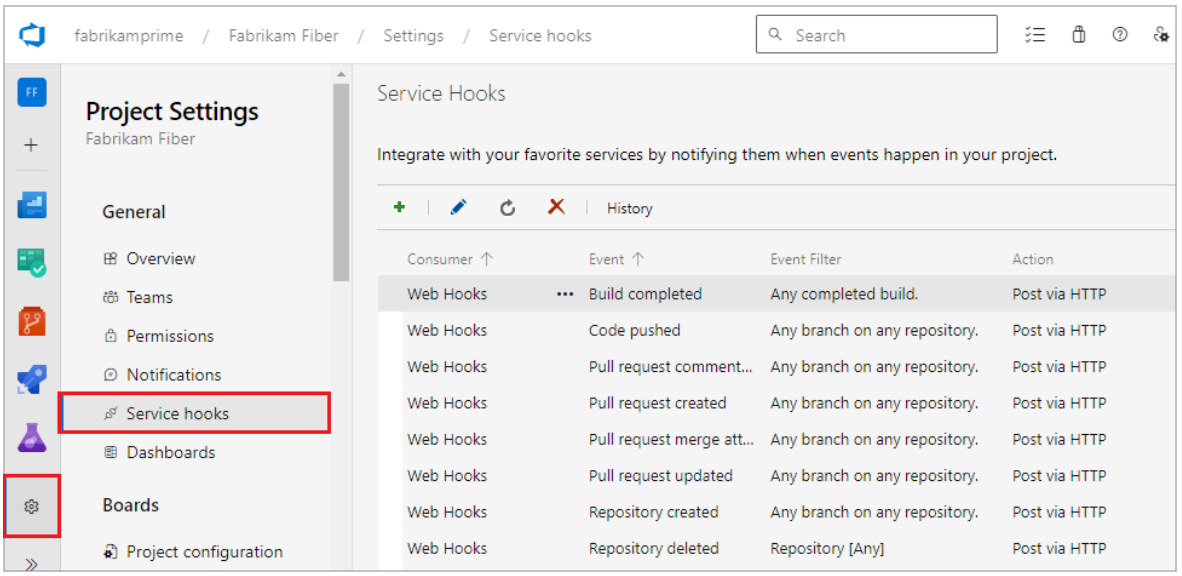
- **Owner** or **Project Collection Administrator** role in the organization.
- A project in the organization, and a service with a public HTTP or HTTPS endpoint that you want to send Azure DevOps events to.

  > **Important**
  >
  > It's recommended to use only HTTPS endpoints. HTTP has the potential to send private data, including authentication headers, unencrypted in the event payload. You must use HTTPS for basic authentication on a webhook.

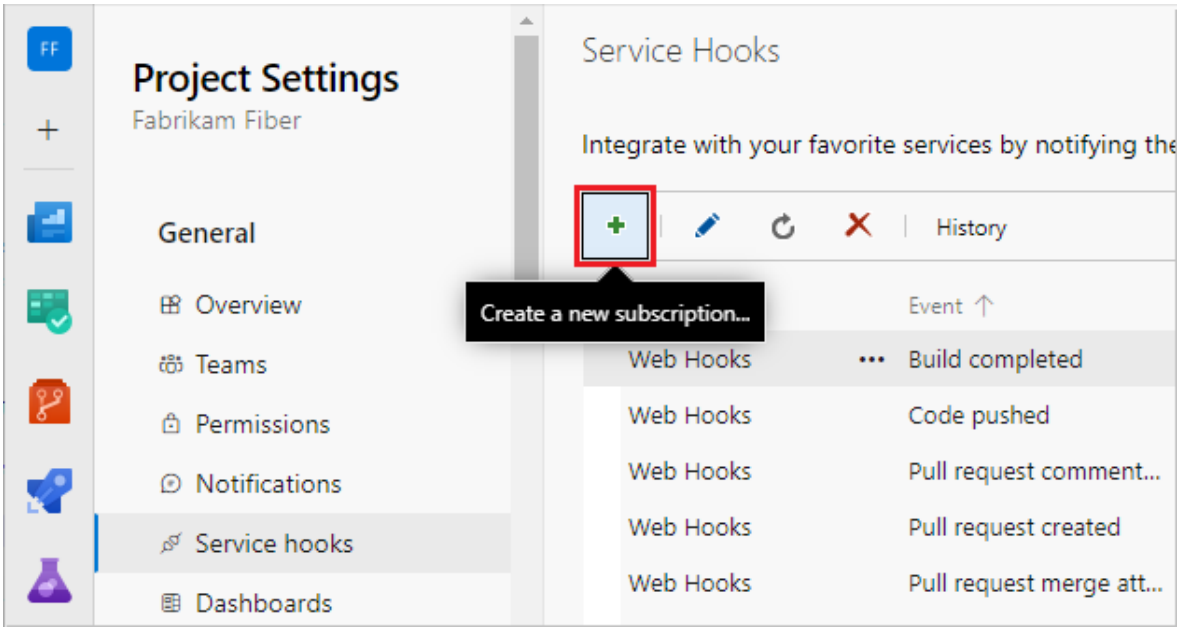
- If connecting to a service behind a virtual private network, ensure that Azure DevOps IP addresses are allowed for inbound connections. See Inbound Connections.

Send JSON representation to a service

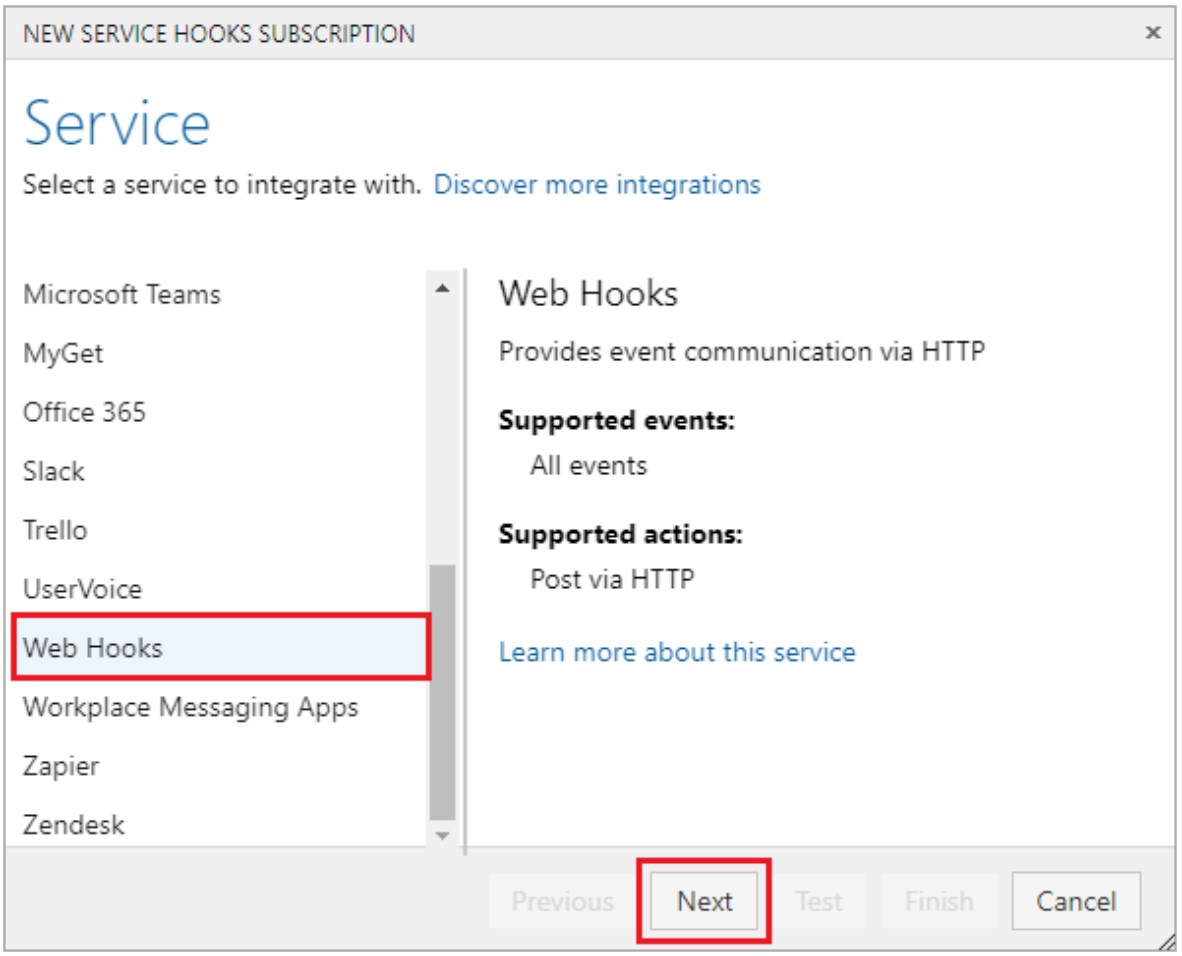
1. In your Azure DevOps project, go to **Project settings** > **Service hooks** at `https://<organization-name>/<project-name>/_settings/serviceHooks`.



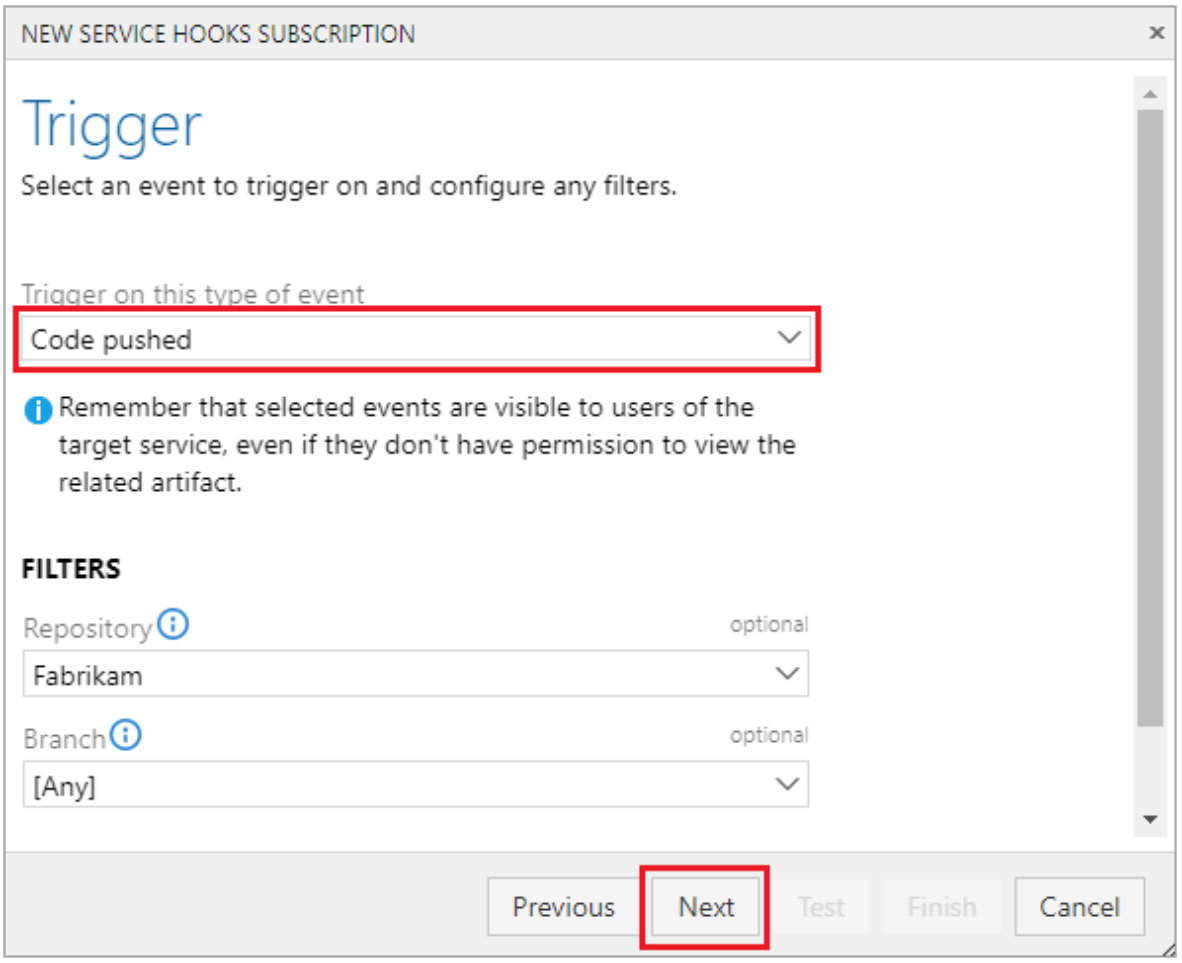
2. On the **Service Hooks** page, select the + icon or **Create subscription**.



3. On the **Service** screen, select **Web Hooks** and then select **Next**.



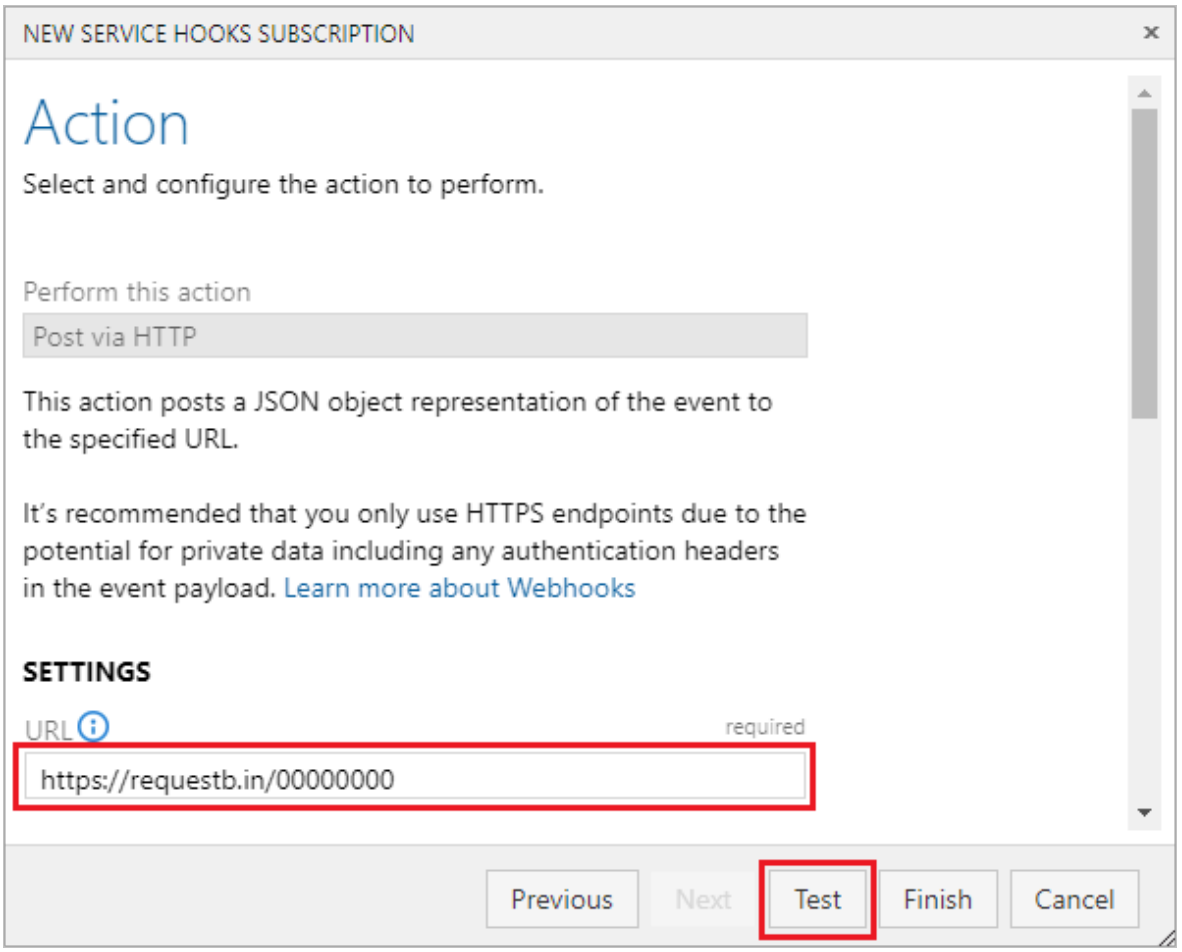
4. On the **Trigger** screen, select and configure the Azure DevOps event you want to trigger the webhook on, and then select **Next**. For more information about the JSON payloads posted by the Azure DevOps consumer, see Service hooks events.



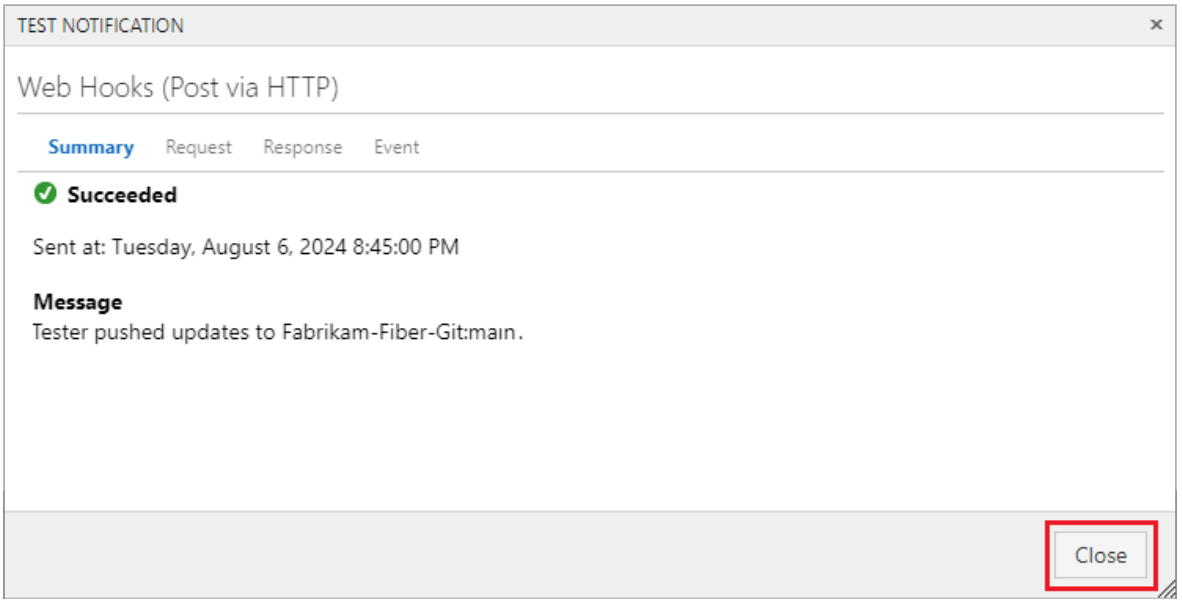
5. On the **Action** screen, configure the target **URL** and what to do when the event occurs. For more information about what to enter in **Resource details to send**, **Messages to send**, and **Detailed messages to send**, see Resource details to send.

> **Note**
>
> Webhooks can't target localhost (loopback) or special range IPv4 /IPv6 addresses.

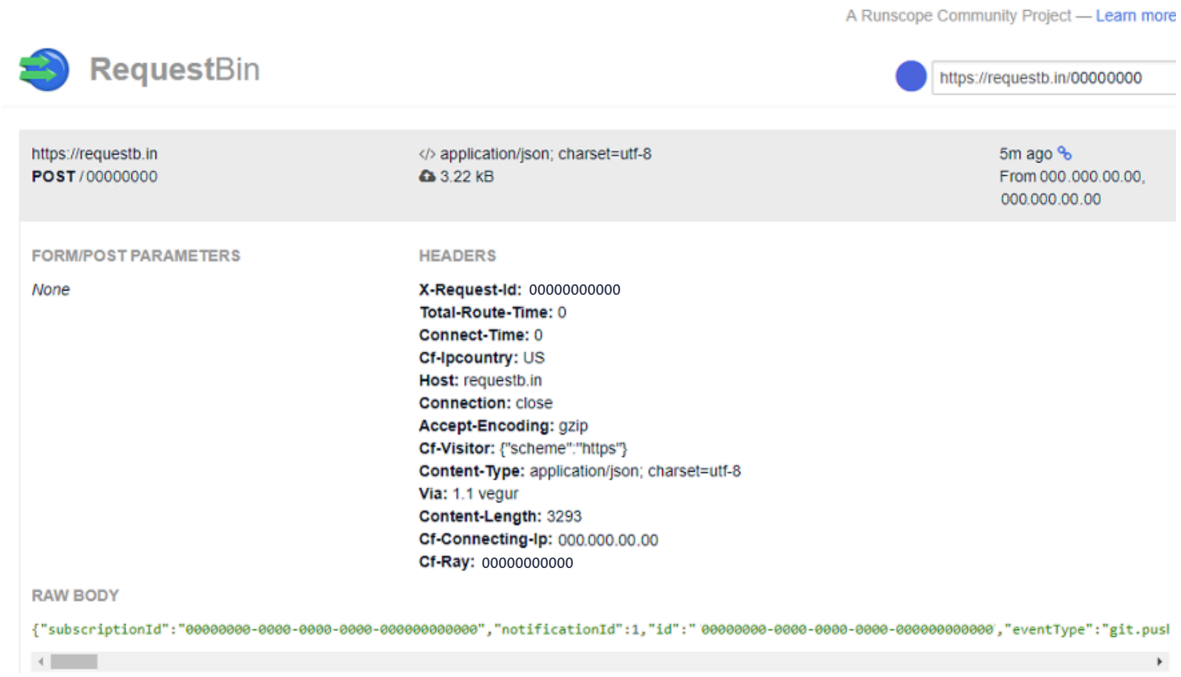
6. Select **Test** to test the service hook subscription.



7. When the test completes successfully, close the notification screen, and then select **Finish** on the **Action** screen.



8. Now the webhook is set up. Go to the target service to view the JSON representation.



> **Tip**
>
> You can also create a webhook programmatically. For more information, see Create a service hook subscription programmatically.

Resource details to send

The **Resource details to send**, **Messages to send**, and **Detailed messages to send** settings in the **Action** pane control the size of the JSON payload to send. **Resource details to send** controls how much of the resource to send. The default is **All**, but you can also choose to send **Minimal**, which sends only key fields like URL and ID, or **None**.

None and **Minimal** are useful in scenarios where the caller doesn't need much or any information about the resource, because it relies on the message or detailed message itself. **None** and **Minimal** are also useful for security reasons. The caller must call back into Azure DevOps Services and go through normal security and permission checks to get more details about the resource.

The following sample JSON shows minimal details about the resource:

JSON

```
{
    "eventType": "git.push",
    ...
    "messages": {
        "text": "...",
        "html": "...",
```

```
            "markdown": "..."
        },
        "detailedMessage": {
            "text": "...",
            "html": "...",
            "markdown": "..."
        },
        "resource": {
            "id": "...",
            "url": "https://...",
            "name": "...",
            "field1:": "..."
        }
    }
```

Related content

- Integrate with service hooks
- Service hooks events
- Create a service hook subscription programmatically

Feedback

Was this page helpful? Yes No

Provide product feedback

Manage authorization of services to access Azure DevOps

Article • 10/17/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

When you use a service that's integrated with Azure DevOps, the industry-standard OAuth 2.0 authorization framework provides safe and secure access to your resources by those other services. With OAuth, you grant a service the authorization to access your Azure DevOps resources, such as work items, source code, and build results.

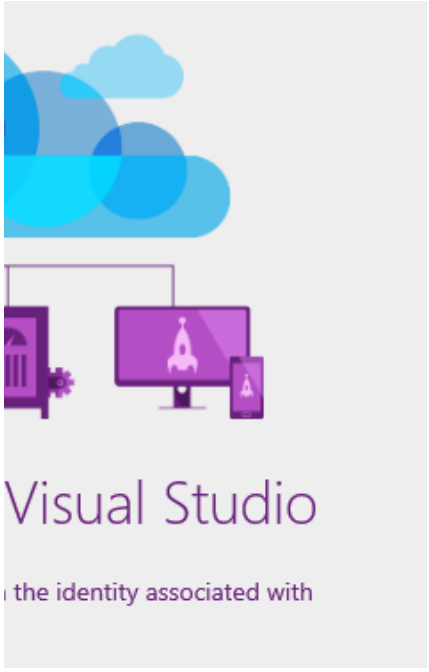
- Authorizations are bound to your credentials, so the service can use the authorization to access your resources in Azure DevOps.
- Use your Microsoft account or your work account to authorize the service.
- The service that you authorize never has access to your Azure DevOps credentials.
- Revoke any authorizations that you've granted to other services.

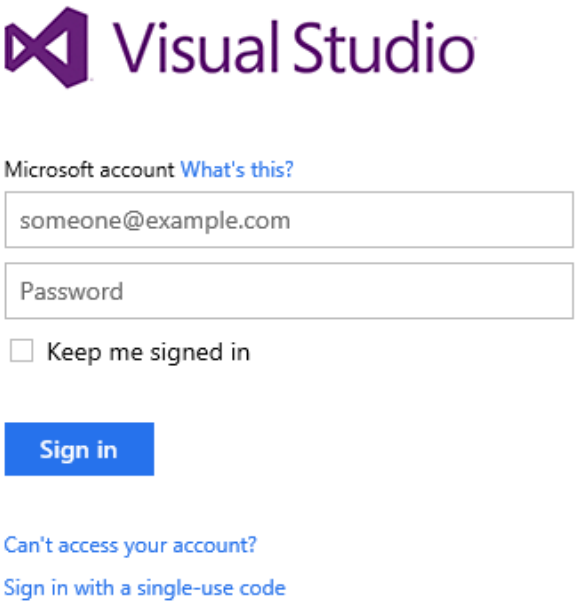
Authorize a service

A typical authorization flow might go like this:

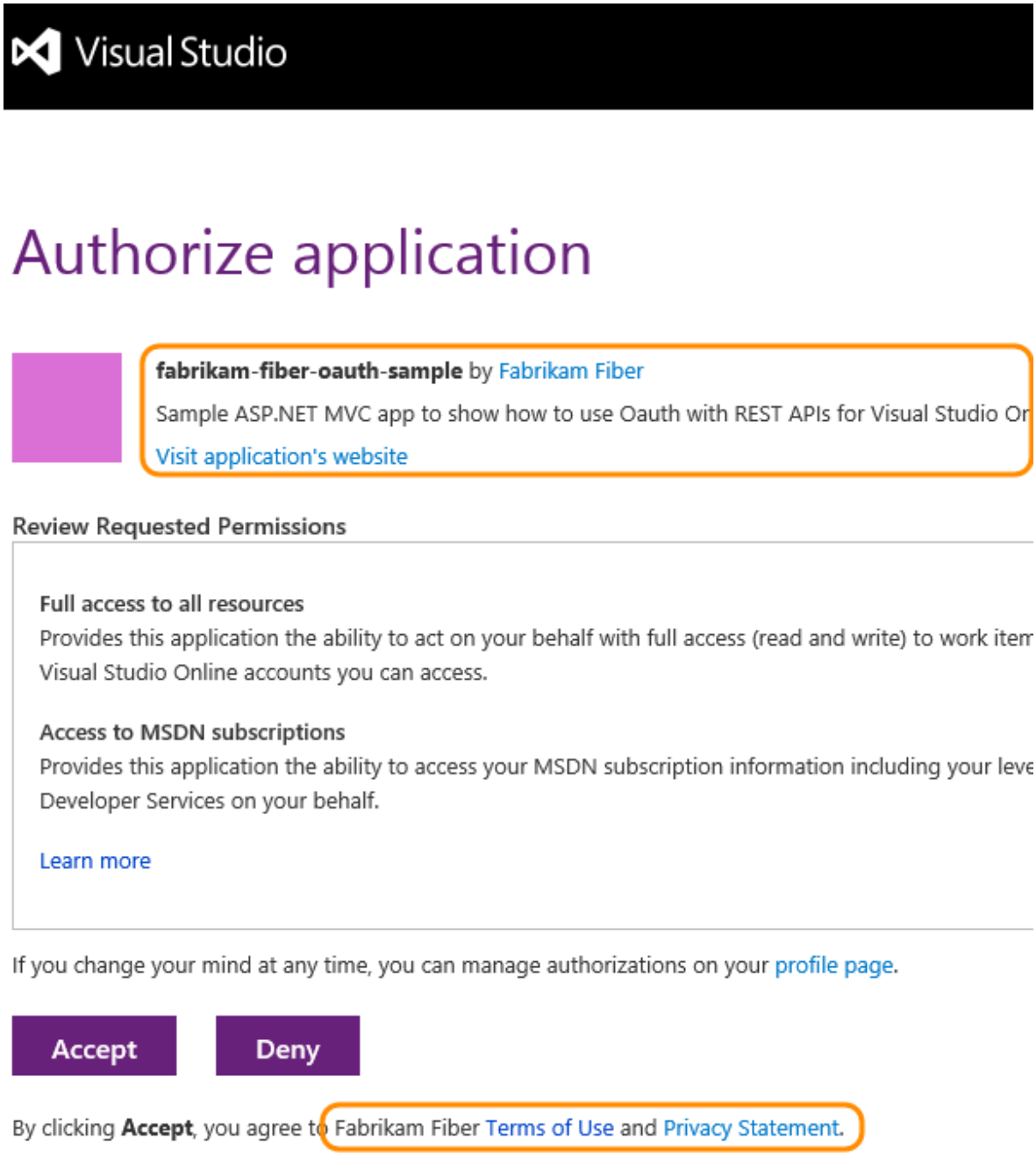
1. You're using a service that uses Azure DevOps resources, so the service requests authorization.

2. If you're not already signed in, Azure DevOps prompts you for your credentials.





3. After you sign in, you get the authorization approval page.



   Services can only request full access to all of the resources that are available to you through the REST APIs, so the authorization request might not be specific.

4. Review the request and approve the authorization.

   The authorized service uses that authorization to access resources in your Visual Studio account.

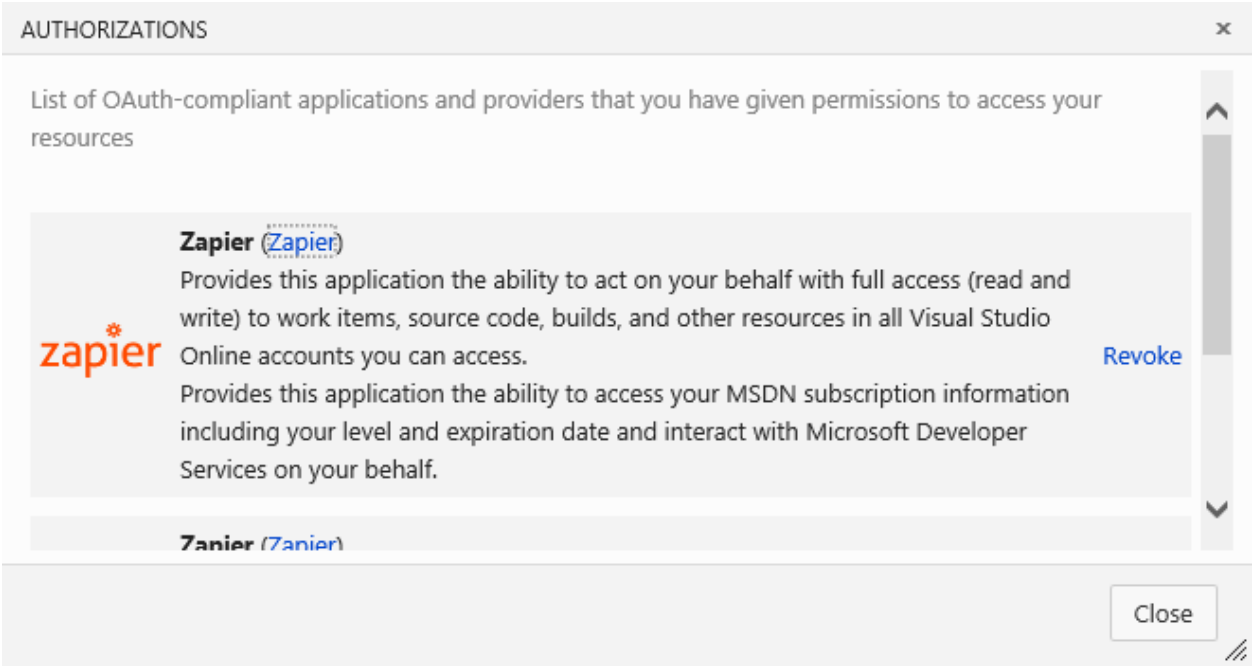
   To ensure an authorization request is legitimate:

- Look for the Azure DevOps branding across the upper portion of the authorization approval page.
- Ensure the authorization approval page URL begins with `https://app.vssps.visualstudio.com/`.

- Pay attention to any HTTPS-related security warnings in your browser.
- Remember that other services don't ask for your credentials directly. They let you provide them to Azure DevOps through the authorization approval page.

Manage authorizations

To see the services that you've authorized to access your account, go to https://app.vssps.visualstudio.com/Profile/View and select **Manage authorizations**.



You can revoke any authorizations here, so the service can't access your account on your behalf.

Service hook consumers

Article • 01/09/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Use service hook consumers to programmatically create a subscription. The subscription specifies the event, the consumer, and the action.

Select the consumer that you want to use in your subscription from the following consumers. For more information about available services, see Integrate with service hooks, Available services.

- Azure Service Bus
- Azure Storage
- Jenkins
- Trello
- Web Hooks
- Zendesk

Azure Service Bus

Provides integration with Microsoft Azure Service Bus, including Notification Hubs.

Send a message to a Notification Hub

This action sends a generic, template notification to the specified Azure Notification Hub. For more information, see the overview.

- Consumer ID: **azureServiceBus**
- Action ID: **serviceBusNotificationHubSend**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**
- Settings:
  - **connectionString**
    - SAS connection string
    - The SAS (shared access signature) connection string to use to connect with Azure Service Bus. This connection string is available in the Azure portal.
    - Data type: **string**
    - Required: **Yes**
  - **notificationHubName**
    - Notification Hub name

        - The name of the notification hub to send the notification to. The name can contain only letters, numbers, periods, hyphens, forward slashes, and underscores. The name must start and end with a letter or number. The hub should already exist.
        - Data type: **string**
        - Required: **Yes**
    - **tagsExpression**
        - Tags
        - The tags expression (for targeting specific sets of devices). Learn more.
        - Data type: **string**
        - Required: **No**

Send a message to a Service Bus Queue

This action sends a JSON string representation of the event to the specified Azure Service Bus queue. For more information, see Service Bus queues, articles, and subscriptions.

- Consumer ID: **azureServiceBus**
- Action ID: **serviceBusQueueSend**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**
- Settings:
    - **connectionString**
        - SAS connection string
        - The SAS (shared access signature) connection string to use to connect with Azure Service Bus. This connection string is available in the Azure portal.
        - Data type: **string**
        - Required: **Yes**
    - **queueName**
        - Queue name
        - The name of the queue to send the message to. The name can contain only letters, numbers, periods, hyphens, forward slashes, and underscores. The name must start and end with a letter or number. If the queue doesn't exist, it's created if the specified connection string has the necessary permissions.
        - Data type: **string**
        - Required: **Yes**
    - **resourceDetailsToSend**
        - Resource details to send
        - Control the resource fields to send
        - Data type: **string**

    - Required: **No**
  - **messagesToSend**
    - Messages to send
    - Control the messages to send
    - Data type: **string**
    - Required: **No**
  - **detailedMessagesToSend**
    - Detailed messages to send
    - Control the detailed messages to send
    - Data type: **string**
    - Required: **No**

Send a message to a Service Bus Topic

This action sends a JSON string representation of the event to the specified Azure Service Bus topic. For more information, see Use the Azure portal to create a Service Bus topic and subscriptions to the topic.

- Consumer ID: **azureServiceBus**
- Action ID: **serviceBusTopicSend**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**
- Settings:
  - **connectionString**
    - SAS connection string
    - The SAS (shared access signature) connection string to use to connect with Azure Service Bus. This connection string is available in the Azure portal.
    - Data type: **string**
    - Required: **Yes**
  - **topicName**
    - Topic name
    - The name of the topic to send the message to. The name can contain only letters, numbers, periods, hyphens, forward slashes, and underscores. The name must start and end with a letter or number. If the topic doesn't exist, it gets created if the specified connection string has the necessary permissions.
    - Data type: **string**
    - Required: **Yes**
  - **resourceDetailsToSend**
    - Resource details to send
    - Control the resource fields to send
    - Data type: **string**

    - Required: **No**
  - **messagesToSend**
    - Messages to send
    - Control the messages to send
    - Data type: **string**
    - Required: **No**
  - **detailedMessagesToSend**
    - Detailed messages to send
    - Control the detailed messages to send
    - Data type: **string**
    - Required: **No**

Azure Storage

Provides integration with Microsoft Azure Storage.

Insert a message in a Storage Queue

This action inserts a JSON string representation of the event to the specified Azure storage queue. For more information, see What are Azure queues?.

- Consumer ID: **azureStorageQueue**
- Action ID: **enqueue**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**
- Settings:
  - **accountName**
    - Storage account name
    - The name associated with your Azure storage account (for example, `https://STORAGE_ACCOUNT_NAME.queue.core.windows.net`).
    - Data type: **string**
    - Required: **Yes**
  - **accountKey**
    - Storage account key
    - The key associated with your Azure storage account.
    - Data type: **string**
    - Required: **Yes**
  - **queueName**
    - Queue name

        - The lowercase-only name of the queue to be used within Azure storage. A queue by this name gets created if it doesn't already exist.
        - Data type: **string**
        - Required: **Yes**
    - **visiTimeout**
        - Message visibility timeout (in seconds)
        - Specifies the visibility timeout value, in seconds, for the enqueued message, relative to server time. The value must be larger than or equal to 0, and can't be larger than seven days, or 604,800 seconds. The visibility timeout must be set to a value smaller than the message's time-to-live value.
        - Data type: **number**
        - Required: **Yes**
    - **ttl**
        - Message time-to-live (in seconds)
        - Specifies the time-to-live interval for the queue message, in seconds. The maximum time-to-live allowed is seven days, or 604,800 seconds.
        - Data type: **number**
        - Required: **Yes**
    - **resourceDetailsToSend**
        - Resource details to send
        - Control the resource fields to send
        - Data type: **string**
        - Required: **No**
    - **messagesToSend**
        - Messages to send
        - Control the messages to send
        - Data type: **string**
        - Required: **No**
    - **detailedMessagesToSend**
        - Detailed messages to send
        - Control the detailed messages to send
        - Data type: **string**
        - Required: **No**

Jenkins

Jenkins is a continuous integration server, which allows building and testing software projects continuously.

Trigger Git build

Triggers a build configured to use a Git repository using the Jenkins Git Plugin.

- Consumer ID: **jenkins**
- Action ID: **triggerGitBuild**
- Supported events: **git.push**
- Settings:
  - **serverBaseUrl**
    - Jenkins base URL
    - The base URL that hosts the Jenkins server
    - Data type: **uri**
    - Required: **Yes**
  - **basicAuthCredentials** We recommend using service principals and managed identities in Azure DevOps.
    - Basic authentication credentials
    - Enter standard HTTP authentication credentials. Basic HTTP authentication send credentials in plain text (unencrypted) which means you should use a URL beginning with "https" to enable encryption of these credentials via secure transport layer (SSL)
    - Data type: **string**
    - Required: **Yes**

Trigger generic build

Triggers a generic Jenkins build, invoking the Jenkins build URL.

- Consumer ID: **jenkins**
- Action ID: **triggerGenericBuild**
- Supported events: **git.push, build.complete, tfvc.checkin**
- Settings:
  - **serverBaseUrl**
    - Jenkins base URL
    - The base URL that hosts the Jenkins server
    - Data type: **uri**
    - Required: **Yes**
  - **basicAuthCredentials** We recommend using service principals and managed identities in Azure DevOps.
    - Basic authentication credentials
    - Enter credentials for standard HTTP authentication. Basic HTTP authentication sends credentials in plain text (unencrypted) which means you should use a URL beginning with "https" to enable encryption of these credentials via secure transport layer (SSL).

    - Data type: **string**
    - Required: **No**
  - **buildName**
    - Build
    - The build name to trigger
    - Data type: **string**
    - Required: **Yes**
  - **buildAuthToken**
    - Build token
    - The authorization token in the form of a string so that only users who know it would be able to remotely trigger this project's builds
    - Data type: **string**
    - Required: **No**
  - **buildParameterized**
    - Accepts parameters
    - Indicates if the build is parameterized or not (build parameters are optionally specified previously)
    - Data type: **boolean**
    - Required: **No**
  - **buildParams**
    - Build parameters
    - Build parameters names and values separated by a colon(for example "param1:value1") with each name-value pair appearing on its own line of text
    - Data type: **string**
    - Required: **No**

Trello

Provides integration with Trello.

Create a card

This action creates a card on an existing list in Trello. A card can represent a task, issue, event, or just about anything. A card's state is typically determined by what list it is on. For more information, see Create a service hook with Trello.

- Consumer ID: **trello**
- Action ID: **createCard**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**

- Settings:
  - **userToken**
    - User token (Need one? Get it now.)
    - Your user token provided by Trello. To learn how to obtain this token, select the link in the previously described action description.
    - Data type: **string**
    - Required: **Yes**
  - **boardId**
    - Board
    - The name of the board on which the Trello card gets created.
    - Data type: **string**
    - Required: **Yes**
  - **listId**
    - List
    - The name of the list on which the Trello card gets created.
    - Data type: **string**
    - Required: **Yes**
  - **labels**
    - Labels
    - A comma-separated list of label colors to apply to the created card. Valid label color names are red, orange, yellow, green, blue, and purple.
    - Data type: **string**
    - Required: **No**
  - **addToTop**
    - Create at beginning of list
    - Indicates if the card should be created at the beginning of the Trello list, instead of the end.
    - Data type: **boolean**
    - Required: **No**

Create a list

This action creates a list on an existing board in Trello. A list is used to organize cards on a board and typically represents a state. For more information, see Create a service hook with Trello.

- Consumer ID: **trello**
- Action ID: **createList**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**
- Settings:

    - **userToken**
        - User token (need one? Get it now .)
        - Your user token provided by Trello. To learn how to obtain this token, select the link in the previously described action description.
        - Data type: **string**
        - Required: **Yes**
    - **boardId**
        - Board
        - The name of the board on which the Trello list gets created.
        - Data type: **string**
        - Required: **Yes**
    - **addToBottom**
        - Create at bottom of board
        - Indicates if the list should be created at the bottom of the board, instead of the top.
        - Data type: **boolean**
        - Required: **No**

Webhooks

Webhooks provide event communication via HTTP.

Post via HTTP

This action posts a JSON object representation of the event to the specified URL. HTTPS endpoints are recommended due to the potential for private data in the event payload. For more information, see Webhooks.

- Consumer ID: **webHooks**
- Action ID: **httpRequest**
- Supported events: **build.complete, git.push, tfvc.checkin, workitem.created, workitem.commented, workitem.updated**
- Settings:
    - **url**
        - URL
        - The URL to which an HTTP POST is sent.
        - Data type: **uri**
        - Required: **Yes**
    - **httpHeaders**
        - HTTP headers

    - HTTP header keys and values separated by a colon (for example "Key1:value1") with each key-value-pair appearing on its own line of text.
    - Data type: **string**
    - Required: **No**
  - **basicAuthCredentials** We recommend using service principals and managed identities in Azure DevOps.
    - Basic authentication credentials
    - Enter credentials for standard HTTP authentication. Basic HTTP authentication sends credentials in plain text (unencrypted) which means you should use a URL beginning with "https" to enable encryption of these credentials via secure transport layer (SSL).
    - Data type: **string**
    - Required: **Yes**
  - **resourceDetailsToSend**
    - Resource details to send
    - Control the resource fields to send
    - Data type: **string**
    - Required: **No**
  - **messagesToSend**
    - Messages to send
    - Control the messages to send
    - Data type: **string**
    - Required: **No**
  - **detailedMessagesToSend**
    - Detailed messages to send
    - Control the detailed messages to send
    - Data type: **string**
    - Required: **No**

Zendesk

Zendesk is a SaaS suite that offers help desk ticketing, issue tracking, and customer service support.

Create a private comment in a ticket

Use Zendesk to create a private comment in a ticket.

- Consumer ID: **zendesk**
- Action ID: **createPrivateComment**

- Supported events: **workitem.commented**
- Settings:
  - **accountName**
    - Account name
    - Zendesk account name like `https://{account name}.zendesk.com`
    - Data type: **string**
    - Required: **Yes**
  - **username**
    - User name
    - The Zendesk user name of a user who updates tickets
    - Data type: **string**
    - Required: **Yes**
  - **apiToken**
    - API token
    - The Zendesk API token (can be found in Zendesk app in Admin > Channels > API)
    - Data type: **string**
    - Required: **Yes**

Related articles

- Authorize other services
- Service hooks overview

Feedback

Was this page helpful? Yes No

Provide product feedback

Service hooks events

Article • 09/18/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Available event types

- **Build and release**
  - Build completed
  - Release created
  - Release abandoned
  - Release deployment approval completed
  - Release deployment approval pending
  - Release deployment completed
  - Release deployment started

- **Pipelines**
  - Check updated
  - Elastic agent pool resized
  - Manual intervention pending
  - Project-level agent pool created
  - Project-level agent pool updated
  - Run state changed
  - Run stage state changed
  - Run stage waiting for approval
  - Run stage approval completed
  - Run job state changed

- **Code**
  - Code checked in
  - Code pushed
  - Pull request created
  - Pull request merge attempted
  - Pull request updated
  - Pull request commented on
  - Repository created
  - Repository deleted
  - Repository forked
  - Repository renamed
  - Repository status changed

- **Work items**
  - Work item commented on
  - Work item created
  - Work item deleted
  - Work item restored
  - Work item updated
- Service connection created
- Service connection updated

> **Note**
>
> The Nuget WebHooks Receivers package provides support for receiving Webhooks from Azure DevOps.

Build and release

Build completed

Event: A build completes.

- Publisher ID: `azure-devops`
- Event ID: `build.complete`
- Resource Name: `build`

Settings

- `definitionName`: Filter events to include only completed builds for the specified pipeline
- `buildStatus`: Filter events to include only completed builds for the specified completion status
  - Valid values:
    - `Succeeded`
    - `PartiallySucceeded`
    - `Failed`
    - `Stopped`

Sample payload

JSON


```
{
  "subscriptionId": "4f6e6328-0251-4814-a009-c01dfa368e3f",
  "notificationId": 1,
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "build.complete",
  "publisherId": "azure-devops",
  "message": {
    "text": "Build 20241202.1 succeeded",
    "html": "Build <a
href=\"https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-
2ebd-42b0-b1e2-257059c4c03d&amp;builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068\">20241202.1</a> succeeded",
    "markdown": "Build [20241202.1]
(https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-2ebd-
42b0-b1e2-257059c4c03d&builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068) succeeded"
  },
  "detailedMessage": {
    "text": "Build 20241202.1 succeeded",
    "html": "Build <a
href=\"https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-
2ebd-42b0-b1e2-257059c4c03d&amp;builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068\">20241202.1</a> succeeded",
    "markdown": "Build [20241202.1]
(https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-2ebd-
42b0-b1e2-257059c4c03d&builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068) succeeded"
  },
  "resource": {
    "id": 2727068,
    "buildNumber": "20241202.1",
    "status": "completed",
    "result": "succeeded",
    "url": "https://dev.azure.com/FabrikamFiber/web/build.aspx?
pcguid=54d02617-2ebd-42b0-b1e2-257059c4c03d&builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068",
    "definition": {
      "id": 1,
      "name": "FabrikamFiber CI"
    },
    "project": {
      "id": "54d02617-2ebd-42b0-b1e2-257059c4c03d",
      "name": "FabrikamFiber"
    }
  },
  "createdDate": "2024-12-02T12:21:13.8866607Z"
}{
  "subscriptionId": "4f6e6328-0251-4814-a009-c01dfa368e3f",
  "notificationId": 1,
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "build.complete",
  "publisherId": "azure-devops",
```

```
  "message": {
    "text": "Build 20241202.1 succeeded",
    "html": "Build <a
href=\"https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-
2ebd-42b0-b1e2-257059c4c03d&amp;builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068\">20241202.1</a> succeeded",
    "markdown": "Build [20241202.1]
(https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-2ebd-
42b0-b1e2-257059c4c03d&builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068) succeeded"
  },
  "detailedMessage": {
    "text": "Build 20241202.1 succeeded",
    "html": "Build <a
href=\"https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-
2ebd-42b0-b1e2-257059c4c03d&amp;builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068\">20241202.1</a> succeeded",
    "markdown": "Build [20241202.1]
(https://dev.azure.com/FabrikamFiber/web/build.aspx?pcguid=54d02617-2ebd-
42b0-b1e2-257059c4c03d&builduri=azure-
devops%3a%2f%2fBuild%2fBuild%2f2727068) succeeded"
  },
  "resource": {
    "_links": {
      "self": {
        "href": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-
0000-000000000000/_apis/build/Builds/2727068"
      },
      "web": {
        "href": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-
0000-000000000000/_build/results?buildId=2727068"
      },
      "sourceVersionDisplayUri": {
        "href": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-
0000-000000000000/_apis/build/builds/2727068/sources"
      },
      "timeline": {
        "href": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-
0000-000000000000/_apis/build/builds/2727068/Timeline"
      },
      "badge": {
        "href": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-
0000-000000000000/_apis/build/status/4658"
      }
    },
    "properties": {},
    "tags": [],
    "validationResults": [],
    "plans": [
      {
        "planId": "cd96240e-881a-49e9-8135-89a4ad458a6d"
      }
    ],
    "triggerInfo": {},
    "id": 2727068,
```

```
    "buildNumber": "20241202.1",
    "status": "completed",
    "result": "succeeded",
    "queueTime": "2024-12-02T12:18:45.7367977Z",
    "startTime": "2024-12-02T12:18:56.6205723Z",
    "finishTime": "2024-12-02T12:21:08.520904Z",
    "url": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-0000-
000000000000/_apis/build/Builds/2727068",
    "definition": {
      "drafts": [],
      "id": 4658,
      "name": "MainRepo (1)",
      "url": "https://dev.azure.com/FabrikamFiber/00000000-0000-0000-0000-
000000000000/_apis/build/Definitions/4658?revision=1",
      "uri": "azure-devops:///Build/Definition/4658",
      "path": "\\",
      "type": "build",
      "queueStatus": "enabled",
      "revision": 1,
      "project": {
        "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-0000-
000000000000",
        "name": "FabrikamFiberChat",
        "url": "https://dev.azure.com/FabrikamFiber/_apis/projects/00000000-
0000-0000-0000-000000000000",
        "state": "wellFormed",
        "revision": 876,
        "visibility": "organization",
        "lastUpdateTime": "2024-04-06T14:51:16.337Z"
      }
    },
    "buildNumberRevision": 1,
    "project": {
      "id": "00000000-0000-0000-0000-000000000000",
      "name": "FabrikamFiberChat",
      "url": "https://dev.azure.com/FabrikamFiber/_apis/projects/00000000-
0000-0000-0000-000000000000",
      "state": "wellFormed",
      "revision": 876,
      "visibility": "organization",
      "lastUpdateTime": "2024-04-06T14:51:16.337Z"
    },
    "uri": "azure-devops:///Build/Build/2727068",
    "sourceBranch": "refs/heads/main",
    "sourceVersion": "00000000-0000-0000-0000-000000000000",
    "queue": {
      "id": 4799,
      "name": "Azure Pipelines",
      "pool": {
        "id": 112,
        "name": "Azure Pipelines",
        "isHosted": true
      }
    },
    "priority": "normal",
```

```
    "reason": "manual",
    "requestedFor": {
      "displayName": "Fabrikam Fiber",
      "url": "https://spsprodwus22.vssps.visualstudio.com/A9985ce4a-0206-
43df-a8ea-076ae68596a9/_apis/Identities/57a5d794-9773-6c32-bbb6-
550585695119",
      "_links": {
        "avatar": {
          "href":
"https://dev.azure.com/FabrikamFiber/_apis/GraphProfile/MemberAvatars/aad.NT
dhNWQ3OTQtOTc3My03YzMyLQJiYjYtNTUwNTg1Njk1MTE5"
        }
      },
      "id": "00000000-0000-0000-0000-000000000000",
      "uniqueName": "chuck@FabrikamFiber.com",
      "imageUrl":
"https://dev.azure.com/FabrikamFiber/_apis/GraphProfile/MemberAvatars/aad.NT
dhNWQ3OTQtOTc1My03YzMyLWJiYjYtNTUwNTg1Njk1MTE5",
      "descriptor": "aad.NTdhNWQ3OTQtOTc6My03YzMyLWJiYjYtNTUwNTg1Njk1MTE5"
    },
    "requestedBy": {
      "displayName": "Chuck Reinhart",
      "url": "https://spsprodwus22.vssps.visualstudio.com/A9985ce4a-0206-
43df-a8ea-076ae68596a9/_apis/Identities/57a5d794-9773-6c32-bbb6-
550585695119",
      "_links": {
        "avatar": {
          "href":
"https://dev.azure.com/FabrikamFiber/_apis/GraphProfile/MemberAvatars/aad.NT
dhNWQ3OTQtOTc3My03YzMyLQJiYjYtNTUwNTg1Njk1MTE5"
        }
      },
      "id": "00000000-0000-0000-0000-000000000000",
      "uniqueName": "chuck@FabrikamFiber.com",
      "imageUrl":
"https://dev.azure.com/FabrikamFiber/_apis/GraphProfile/MemberAvatars/aad.NT
dhNWQ3OTQtOTc1My03YzMyLWJiYjYtNTUwNTg1Njk1MTE5",
      "descriptor": "aad.NTdhNWQ3OTQtOTc6My03YzMyLWJiYjYtNTUwNTg1Njk1MTE5"
    },
    "lastChangedDate": "2024-12-02T12:21:08.96Z",
    "lastChangedBy": {
      "displayName": "Microsoft.VisualStudio.Services.TFS",
      "url": "https://spsprodwus22.vssps.visualstudio.com/A9982ce4a-0206-
43df-a8ea-076ae68596a9/_apis/Identities/00000000-0000-0000-0000-
000000000000",
      "_links": {
        "avatar": {
          "href":
"https://dev.azure.com/FabrikamFiber/_apis/GraphProfile/MemberAvatars/s2s.MD
AwMDAwMDItMDAwMC02ODg4LTgwMDAtMDAwMDAwMDAwMDAwQDJjODk1OTA4LTA0ZTAtNDk1Mi04OW
ZkLTU0YjAwNDZkNjI4OA"
        }
      },
      "id": "00000000-0000-0000-0000-0000000000000-0000-0000-0000-
000000000000",
```

```
      "uniqueName": "00000000-0000-0000-0000-000000000000@00000000-0000-
0000-0000-000000000000",
      "imageUrl":
"https://dev.azure.com/FabrikamFiber/_apis/GraphProfile/MemberAvatars/s2s.MD
AwMDAwMDItMDAwMC04ODg4LTgwMDAtMDAwMDAwMDAwMDAwQDJjODk1OTA4LTA2ZTAtNDk1Mi04OW
ZkLTU0YjAwNDZkNjI4OA",
      "descriptor":
"s2s.MDAwMDAwMDItMDAwMC04ODg4LTgwMDAtMDAwMDAwMDAwMDAwQDJjODk1OTA4LTA2ZTAtNDk
1Mi04OWZkLTU0YjAwNDZkNjI4OA"
    },
    "orchestrationPlan": {
      "planId": "00000000-0000-0000-0000-000000000000"
    },
    "logs": {
      "id": 0,
      "type": "Container",
      "url":
"https://dev.azure.com/FabrikamFiber//_apis/build/builds/00000000-0000-0000-
0000-000000000000/logs"
    },
    "repository": {
      "id": "00000000-0000-0000-0000-000000000000",
      "type": "TfsGit",
      "name": "MainRepo",
      "url":
"https://dev.azure.com/FabrikamFiber/FabrikamFiberChat/_git/FabrikamFiberCha
t",
      "clean": null,
      "checkoutSubmodules": false
    },
    "retainedByRelease": false,
    "triggeredByBuild": null,
    "appendCommitMessageToRunName": true
  },
  "resourceVersion": "2.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000",
      "baseUrl": "https://dev.azure.com/FabrikamFiber/"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000",
      "baseUrl": "https://dev.azure.com/FabrikamFiber/"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000",
      "baseUrl": "https://dev.azure.com/FabrikamFiber/"
    }
  },
  "createdDate": "2024-12-02T12:21:13.8866607Z"
}
```


Release abandoned

Event: A release is abandoned.

- Publisher ID: `rm`
- Event ID: `ms.azure-devops-release.release-abandoned-event`
- Resource Name: `resource`

Settings

- `releaseDefinitionId`: Filter events to include only completed deployments for the specified pipeline

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "ms.azure-devops-release.release-abandoned-event",
  "publisherId": "rm",
  "scope": "all",
  "resource": {
    "release": {
      "id": "release-id",
      "name": "release-name",
      "status": "abandoned",
      "releaseDefinition": {
        "id": "release-definition-id",
        "name": "release-definition-name"
      },
      "project": {
        "id": "project-id",
        "name": "project-name"
      }
    }
  },
  "createdDate": "2024-12-02T12:21:13.8866607Z"
}
```


Release created

Event: A release is created.

- Publisher ID: `rm`
- Event ID: `ms.azure-devops-release.release-created-event`
- Resource Name: `resource`

Settings

- `releaseDefinitionId`: Filter events to include only completed deployments for the specified pipeline

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000d4d69db4-18d4-413e-bc43-
07f56b531160",
  "eventType": "ms.azure-devops-release.release-created-event",
  "publisherId": "rm",
  "scope": "all",
  "message": {
    "text": "Release Release-1 created.",
    "html": "<a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/5'>Release-1</a> created.",
    "markdown": "Release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/5) created."
  },
  "detailedMessage": {
    "text": "Release Release-1 created from release pipeline
Fabrikam.CD.\\r\\nRelease description: QFE release for fixing
title\\r\\nContinuousIntegration Requested for Chuck Reinhart\\r\\n- Build:
fabrikam.Bd.2016.04.10 & 2 more<\\li>",
    "html": "Release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/5'>Release-1</a> created from release pipeline <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releasedefinitions/1'>Fabrikam.CD</a>.\\r\\n- Release
description: QFE release for fixing title</br>\\r\\n- ContinuousIntegration
Requested for Chuck Reinhart</br>\\r\\n- Build: fabrikam.Bd.2016.04.10 & 2
more<\\li>",
    "markdown": "Release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/5) created from release pipeline [Fabrikam.CD]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releasedefinitions/1).\\r\\n- Release description: QFE
release for fixing title</br>\\r\\n- ContinuousIntegrationRequested for
Chuck Reinhart</br>\\r\\n- Build: fabrikam.Bd.2016.04.10 & 2 more<\\li>"
  },
  "resource": {
    "release": {
      "id": 4,
      "name": "Release-1",
      "status": "active",
      "createdOn": "2016-01-21T08:19:17.26Z",
      "modifiedOn": "2016-01-21T08:19:17.26Z",
```

```
      "modifiedBy": {
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "Chuck Reinhart"
      },
      "createdBy": {
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "Chuck Reinhart"
      },
      "environments": [
        {
          "id": 5,
          "releaseId": 0,
          "name": "Dev",
          "status": "succeeded",
          "variables": {},
          "preDeployApprovals": [],
          "postDeployApprovals": [],
          "preApprovalsSnapshot": {
            "approvals": [],
            "approvalOptions": {
              "requiredApproverCount": 0,
              "releaseCreatorCanBeApprover": true
            }
          },
          "postApprovalsSnapshot": {
            "approvals": []
          },
          "deploySteps": [],
          "rank": 1,
          "definitionEnvironmentId": 1,
          "queueId": 1,
          "environmentOptions": {
            "emailNotificationType": "OnlyOnFailure",
            "emailRecipients": "release.environment.owner;release.creator",
            "skipArtifactsDownload": false,
            "timeoutInMinutes": 0,
            "enableAccessToken": false
          },
          "demands": [],
          "conditions": [],
          "modifiedOn": "2016-01-21T08:19:17.26Z",
          "workflowTasks": [
            {
              "taskId": "00000000-0000-0000-0000-000000000000",
              "version": "*",
              "name": "Deploy Website to Azure",
              "enabled": true,
              "alwaysRun": false,
              "continueOnError": false,
              "timeoutInMinutes": 0,
              "definitionType": null,
              "inputs": {
                "ConnectedServiceName": "b460b0f8-fe23-4dc2-a99c-
fd8b0633fe1c",
                "WebSiteName": "$(webAppName)",
```

```
                "WebSiteLocation": "Southeast Asia",
                "Slot": "",
                "Package": "$(System.DefaultWorkingDirectory)\\**\\*.zip"
              }
            }
          ],
          "deployPhasesSnapshot": [],
          "owner": {
            "id": "00000000-0000-0000-0000-000000000000",
            "displayName": "Chuck Reinhart"
          },
          "scheduledDeploymentTime": "2016-01-21T08:19:17.26Z",
          "schedules": [],
          "release": {
            "id": 5,
            "name": "Release-1",
            "url":
"http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/5"
          }
        }
      ],
      "variables": {},
      "artifacts": [
        {
          "sourceId": "31419848-1780-4137-b7e3-62092e986fd6:1",
          "type": "Build",
          "alias": "Fabrikam.CI",
          "definitionReference": {
            "Definition": {
              "id": "00000000-0000-0000-0000-0000000000001",
              "name": "Fabrikam.CI"
            },
            "Project": {
              "id": "00000000-0000-0000-0000-000000000000031419848-1780-4137-
b7e3-62092e986fd6",
              "name": "Fabrikam"
            }
          },
          "isPrimary": true
        }
      ],
      "releaseDefinition": {
        "id": 1,
        "name": "Fabrikam.CD",
        "url": "http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-
Fiber-Git/_apis/Release/definitions/1"
      },
      "description": "QFE release for fixing title",
      "reason": "continuousIntegration",
      "releaseNameFormat": "Release-$(rev:r)",
      "keepForever": false,
      "definitionSnapshotRevision": 0,
      "comment": "",
      "logsContainerUrl": null,
```

```
      "_links": {}
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-0000-
000000000000",
      "name": "Fabrikam"
    }
  },
  "resourceVersion": "3.0-preview.1",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:27.6570261Z"
}
```


Release deployment approval completed

Event: A deployment approval is completed.

- Publisher ID: `rm`
- Event ID: `ms.azure-devops-release.deployment-approval-completed-event`
- Resource Name: `resource`

Settings

- `releaseApprovalStatus`: Filter events to include only deployments with an approval of the specified status
  - Valid values:
    - `2` - Approved
    - `4` - Rejected
- `releaseApprovalType`: Filter events to include only deployments requesting an approval of the specified type
  - Valid values:
    - `1` - Predeployment
    - `2` - Post-deployment
- `releaseEnvironmentId`: Filter events to include only completed deployments for the specified environment
- `releaseDefinitionId`: Filter events to include only completed deployments for the specified definition

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000106acb39-c61e-4efd-995e-
a9f5e71ba3cd",
  "eventType": "ms.azure-devops-release.deployment-approval-completed-
event",
  "publisherId": "rm",
  "scope": "all",
  "message": {
    "text": "Pre Deployment approval for deployment of release Release-1 on
environment Dev Succeeded.",
    "html": "Pre Deployment approval for release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1'>Release-1</a> on environment <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/definitions/1'>Dev</a> Succeeded.",
    "markdown": "Pre Deployment approval for deployment of release [Release-
1](http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1) on environment [Dev]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/definitions/1) Succeeded."
  },
  "detailedMessage": {
    "text": "Pre Deployment approval for release Release-1 on environment
Dev Succeeded.\r\nApprover: Chuck Reinhart\r\nComment: Approving",
    "html": "Pre Deployment approval for release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1'>Release-1</a> on environment <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/definitions/1'>Dev</a> Succeeded.<br>Approver: Chuck
Reinhart<br>Comment: Approving",
    "markdown": "Pre Deployment approval for release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1) on environment [Dev]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/definitions/1) Succeeded.\r\nApprover: Chuck
Reinhart\r\nComment: Approving"
  },
  "resource": {
    "approval": {
      "id": 1,
      "revision": 1,
      "approvalType": "preDeploy",
      "status": "approved",
      "createdOn": "2024-12-02T12:21:13.8866607Z",
      "modifiedOn": "2024-12-02T12:21:13.8866607Z",
      "comments": "Approving",
      "isAutomated": false,
      "isNotificationOn": false,
      "trialNumber": 1,
      "attempt": 1,
```

```
      "approver": {
        "displayName": "Chuck Reinhart",
        "id": "00000000-0000-0000-0000-000000000000"
      }
    },
    "environment": {
      "id": 1,
      "name": "Dev"
    },
    "release": {
      "id": 1,
      "name": "Release-1",
      "releaseDefinition": {
        "id": 1,
        "name": "Release Definition"
      },
      "project": {
        "id": "project-id",
        "name": "project-name"
      }
    }
  },
  "createdDate": "2024-12-02T12:21:13.8866607Z"
}
```


Release deployment approval pending

Event: A deployment approval is requested.

- Publisher ID: `rm`
- Event ID: `ms.azure-devops-release.deployment-approval-pending-event`
- Resource Name: `resource`

Settings

- `releaseApprovalType`: Filter events to include only deployments requesting an approval of the specified type
  - Valid values:
    - `1` - Predeployment
    - `2` - Post-deployment
- `releaseEnvironmentId`: Filter events to include only completed deployments for the specified environment
- `releaseDefinitionId`: Filter events to include only completed deployments for the specified pipeline

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000a73e7272-e96d-4249-93ac-
7404eacd6801",
  "eventType": "ms.azure-devops-release.deployment-approval-pending-event",
  "publisherId": "rm",
  "scope": "all",
  "message": {
    "text": "Pre deployment approval pending for release Release-1 on
environment Dev.",
    "html": "Pre deployment approval pending for release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1'>Release-1</a> on environment <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1'>Dev</a>.",
    "markdown": "Pre deployment approval pending for release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1) on environment [Dev]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1)."
  },
  "detailedMessage": {
    "text": "Pre deployment approval pending for release Release-1 on
environment Dev.\r\nPending on: Chuck Reinhart\r\nPending since: 09 May 2016
12:09:29 (UTC)",
    "html": "Pre deployment approval pending for release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1'>Release-1</a> on environment <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1'>Dev</a>.
<br>Pending on: Chuck Reinhart<br>Pending since: 09 May 2016 12:09:29
(UTC)",
    "markdown": "Pre deployment approval pending for release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1) on environment [Dev]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1).\r\nPending
on: Chuck Reinhart\r\nPending since: 09 May 2016 12:09:29 (UTC)"
  },
  "resource": {
    "approval": {
      "id": 1,
      "revision": 1,
      "approvalType": "preDeploy",
      "status": "pending",
      "createdOn": "2016-05-09T12:09:29Z",
      "modifiedOn": "2016-05-09T12:09:29Z",
```

```
      "isAutomated": false,
      "isNotificationOn": false,
      "trialNumber": 1,
      "attempt": 1,
      "approver": {
        "displayName": "Chuck Reinhart",
        "id": "00000000-0000-0000-0000-000000000000"
      }
    },
    "environment": {
      "id": 8,
      "name": "Dev"
    },
    "release": {
      "id": 1,
      "name": "Release-1",
      "releaseDefinition": {
        "id": 1,
        "name": "Release Definition"
      },
      "project": {
        "id": "project-id",
        "name": "project-name"
      }
    }
  },
  "createdDate": "2016-05-09T12:09:29Z"
}{
  "id": "00000000-0000-0000-0000-000000000000a73e7272-e96d-4249-93ac-
7404eacd6801",
  "eventType": "ms.azure-devops-release.deployment-approval-pending-event",
  "publisherId": "rm",
  "scope": "all",
  "message": {
    "text": "Pre deployment approval pending for release Release-1 on
environment Dev.",
    "html": "Pre deployment approval pending for release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1'>Release-1</a> on environment <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1'>Dev</a>.",
    "markdown": "Pre deployment approval pending for release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1) on environment [Dev]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1)."
  },
  "detailedMessage": {
    "text": "Pre deployment approval pending for release Release-1 on
environment Dev.\\r\\nPending on: Chuck Reinhart\\r\\nPending since: 09 May
2016 12:09:29 (UTC)",
    "html": "Pre deployment approval pending of release <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
```

```
Git/_apis/Release/releases/1'>Release-1</a> on environment <a
href='http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-
summary&definitionEnvironmentId=8&definitionId=1'>Dev</a>.\\r\\nPending on:
Chuck Reinhart\\r\\nPending  since: 09 May 2016 12:09:29 (UTC)",
    "markdown": "Pre deployment approval pending for release [Release-1]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1) on environment [Dev]
(http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-
summary&definitionEnvironmentId=8&definitionId=1).\\r\\nPending on: Chuck
Reinhart\\r\\nPending  since: 09 May 2016 12:09:29 (UTC)"
  },
  "resource": {
    "approval": {
      "id": 0,
      "revision": 0,
      "approver": {
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "Chuck Reinhart"
      },
      "approvalType": "preDeploy",
      "createdOn": "2016-01-21T08:19:17.26Z",
      "modifiedOn": "2016-01-21T08:19:17.26Z",
      "status": "pending",
      "comments": "",
      "isAutomated": false,
      "isNotificationOn": true,
      "trialNumber": 1,
      "attempt": 0,
      "rank": 1,
      "release": {
        "id": 1,
        "name": "Release-1"
      },
      "releaseDefinition": {
        "id": 1,
        "name": "Fabrikam.CD",
        "url": "http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-
Fiber-Git/_apis/Release/definitions/1"
      },
      "releaseEnvironment": {
        "id": 8,
        "name": "Dev"
      }
    },
    "release": {
      "id": 1,
      "name": "Release-1",
      "status": "active",
      "createdOn": "2016-01-21T08:19:17.26Z",
      "modifiedOn": "2016-01-21T08:19:17.26Z",
      "modifiedBy": {
```

```
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "Chuck Reinhart"
      },
      "createdBy": {
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "Chuck Reinhart"
      },
      "environments": [
        {
          "id": 5,
          "releaseId": 0,
          "name": "Dev",
          "status": "succeeded",
          "variables": {},
          "preDeployApprovals": [],
          "postDeployApprovals": [],
          "preApprovalsSnapshot": {
            "approvals": [],
            "approvalOptions": {
              "requiredApproverCount": 0,
              "releaseCreatorCanBeApprover": true
            }
          },
          "postApprovalsSnapshot": {
            "approvals": []
          },
          "deploySteps": [],
          "rank": 1,
          "definitionEnvironmentId": 1,
          "queueId": 1,
          "environmentOptions": {
            "emailNotificationType": "OnlyOnFailure",
            "emailRecipients": "release.environment.owner;release.creator",
            "skipArtifactsDownload": false,
            "timeoutInMinutes": 0,
            "enableAccessToken": false
          },
          "demands": [],
          "conditions": [],
          "modifiedOn": "2016-01-21T08:19:17.26Z",
          "workflowTasks": [
            {
              "taskId": "00000000-0000-0000-0000-000000000000",
              "version": "*",
              "name": "Deploy Website to Azure",
              "enabled": true,
              "alwaysRun": false,
              "continueOnError": false,
              "timeoutInMinutes": 0,
              "definitionType": null,
              "inputs": {
                "ConnectedServiceName": "b460b0f8-fe23-4dc2-a99c-
fd8b0633fe1c",
                "WebSiteName": "$(webAppName)",
                "WebSiteLocation": "Southeast Asia",
```

```
                "Slot": "",
                "Package": "$(System.DefaultWorkingDirectory)\\**\\*.zip"
              }
            }
          ],
          "deployPhasesSnapshot": [],
          "owner": {
            "id": "00000000-0000-0000-0000-000000000000",
            "displayName": "Chuck Reinhart"
          },
          "scheduledDeploymentTime": "2016-01-21T08:19:17.26Z",
          "schedules": [],
          "release": {
            "id": 1,
            "name": "Release-1",
            "url":
"http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-Fiber-
Git/_apis/Release/releases/1"
          }
        }
      ],
      "variables": {},
      "artifacts": [
        {
          "sourceId": "31419848-1780-4137-b7e3-62092e986fd6:1",
          "type": "Build",
          "alias": "Fabrikam.CI",
          "definitionReference": {
            "Definition": {
              "id": "00000000-0000-0000-0000-0000000000001",
              "name": "Fabrikam.CI"
            },
            "Project": {
              "id": "00000000-0000-0000-0000-000000000000031419848-1780-4137-
b7e3-62092e986fd6",
              "name": "Fabrikam"
            }
          },
          "isPrimary": true
        }
      ],
      "releaseDefinition": {
        "id": 1,
        "name": "Fabrikam.CD",
        "url": "http://dev.azure.com/fabfiber/DefaultCollection/Fabrikam-
Fiber-Git/_apis/Release/definitions/1"
      },
      "description": "QFE release for fixing title",
      "reason": "continuousIntegration",
      "releaseNameFormat": "Release-$(rev:r)",
      "keepForever": false,
      "definitionSnapshotRevision": 0,
      "comment": "",
      "logsContainerUrl": null,
      "_links": {}
```

```
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000000000-0000-0000-0000-
000000000000",
      "name": "Fabrikam"
    }
  },
  "resourceVersion": "3.0-preview.1",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:28.0320509Z"
}
```


Release deployment completed

Event: A deployment completed.

- Publisher ID: `rm`
- Event ID: `ms.azure-devops-release.deployment-completed-event`
- Resource Name: `resource`

Settings

- `releaseEnvironmentId`: Filter events to include only completed deployments for the specified environment
- `releaseDefinitionId`: Filter events to include only completed deployments for the specified pipeline
- `releaseEnvironmentStatus`: Filter events to include only completed deployments with the specified status
  - Valid values:
    - `8` - Canceled
    - `16` - Rejected
    - `4` - Succeeded
    - `128` - Partially Succeeded

Sample payload

JSON

```
{
  "id": "00000000-0000-0000-0000-000000000000c3e52c57-187a-45c4-abe2-
184a48291bad",
  "eventType": "ms.azure-devops-release.deployment-completed-event",
  "publisherId": "rm",
  "scope": "all",
  "message": {
    "text": "Deployment of release Release-1 on environment Dev Succeeded.",
    "html": "Deployment on environment <a
href='http://fabfiber.visualstudio.com/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1'>Dev</a>
Succeeded.",
    "markdown": "Deployment on environment [Dev]
(http://fabfiber.visualstudio.com/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1) Succeeded."
  },
  "detailedMessage": {
    "text": "Deployment of release Release-1 on environment Dev Succeeded.
Time to deploy: 0.11 minutes.",
    "html": "Deployment on environment <a
href='http://fabfiber.visualstudio.com/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1'>Dev</a>
Succeeded. Time to deploy: 0.11 minutes.",
    "markdown": "Deployment on environment [Dev]
(http://fabfiber.visualstudio.com/DefaultCollection/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=8&definitionId=1) Succeeded.
Time to deploy: 0.11 minutes."
  },
  "resource": {
    "deployment": {
      "id": 1,
      "status": "succeeded",
      "release": {
        "id": 1,
        "name": "Release-1",
        "releaseDefinition": {
          "id": 1,
          "name": "Release Definition"
        },
        "project": {
          "id": "project-id",
          "name": "project-name"
        }
      },
      "environment": {
        "id": 8,
        "name": "Dev"
      }
    }
  },
```

```
  "createdDate": "2024-12-02T12:21:13.8866607Z"
}
```


Release deployment started

Event: A deployment started.

- Publisher ID: `rm`
- Event ID: `ms.azure-devops-release.deployment-started-event`
- Resource Name: `resource`

Settings

- `releaseEnvironmentId`: Filter events to include only completed deployments for the specified environment
- `releaseDefinitionId`: Filter events to include only completed deployments for the specified definition

Sample payload

JSON


```
{
    "id": "00000000-0000-0000-0000-000000000001f04688d-98bb-4206-850f-
43389f4c8cb4",
    "eventType": "ms.azure-devops-release.deployment-started-event",
    "publisherId": "rm",
    "message": {
        "text": "Deployment of release Release-5 to stage Dev started.",
        "html": "Deployment on stage <a
href='http://fabfiber.visualstudio.com/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=1&definitionId=4'>Dev</a>
started.",
        "markdown": "Deployment on stage [Dev]
(https://fabfiber.visualstudio.com/Fabrikam-Fiber-Git/_apps/hub/ms.azure-
devops-releaseManagement-web.hub-explorer?_a=environment-
summary&definitionEnvironmentId=1&definitionId=4) started."
    },
    "detailedMessage": {
        "text": "Deployment of release Release-5 on stage Dev
started.\r\nTrigger: Manual",
        "html": "Deployment on stage <a
href='Dev'>http://fabfiber.visualstudio.com/Fabrikam-Fiber-
Git/_apps/hub/ms.azure-devops-releaseManagement-web.hub-explorer?
_a=environment-summary&definitionEnvironmentId=1&definitionId=4</a> started.
<br>Trigger: Manual",
```

```
        "markdown": "Deployment on stage [Release-1]
(https://fabfiber.visualstudio.com/Fabrikam-Fiber-Git/_apps/hub/ms.azure-
devops-releaseManagement-web.hub-explorer?_a=environment-
summary&definitionEnvironmentId=1&definitionId=4) started.\r\nTrigger: Dev"
    },
    "resource": {
        "environment": {
            "id": 5,
            "releaseId": 0,
            "name": "Dev",
            "status": "queued",
            "variables": {},
            "variableGroups": [],
            "preDeployApprovals": [],
            "postDeployApprovals": [],
            "preApprovalsSnapshot": {
                "approvals": [],
                "approvalOptions": {
                    "requiredApproverCount": 0,
                    "releaseCreatorCanBeApprover": true,

"autoTriggeredAndPreviousEnvironmentApprovedCanBeSkipped": false,
                    "enforceIdentityRevalidation": false,
                    "timeoutInMinutes": 0,
                    "executionOrder": "beforeGates"
                }
            },
            "postApprovalsSnapshot": {
                "approvals": []
            },
            "deploySteps": [],
            "rank": 1,
            "definitionEnvironmentId": 1,
            "queueId": 1,
            "environmentOptions": {
                "emailNotificationType": "OnlyOnFailure",
                "emailRecipients":
"release.environment.owner;release.creator",
                "skipArtifactsDownload": false,
                "timeoutInMinutes": 0,
                "enableAccessToken": false,
                "publishDeploymentStatus": false,
                "badgeEnabled": false,
                "autoLinkWorkItems": false,
                "pullRequestDeploymentEnabled": false
            },
            "demands": [],
            "conditions": [],
            "modifiedOn": "2016-01-21T08:19:17.26Z",
            "workflowTasks": [],
            "deployPhasesSnapshot": [],
            "owner": {
                "displayName": "Chuck Reinhart",
                "id": "00000000-0000-0000-0000-000000000000"
            },
```

```
            "scheduledDeploymentTime": "2016-01-21T08:19:17.26Z",
            "schedules": [],
            "release": {
                "id": 5,
                "name": "Release-5",
                "_links": {
                    "web": {
                        "href": "https://fabfiber.visualstudio.com/Fabrikam-
Fiber-Git/_release?releaseId=1&_a=release-summary"
                    }
                }
            },
            "preDeploymentGatesSnapshot": {
                "id": 0,
                "gatesOptions": null,
                "gates": []
            },
            "postDeploymentGatesSnapshot": {
                "id": 0,
                "gatesOptions": null,
                "gates": []
            }
        },
        "release": {
            "id": 0,
            "name": null,
            "status": "undefined",
            "createdOn": "0001-01-01T00:00:00",
            "modifiedOn": "0001-01-01T00:00:00",
            "modifiedBy": null,
            "createdBy": null,
            "environments": [],
            "variables": {},
            "variableGroups": [],
            "artifacts": [],
            "releaseDefinition": {
                "id": 1,
                "name": "Fabrikam.CD",
                "projectReference": null,
                "_links": {}
            },
            "releaseDefinitionRevision": 0,
            "reason": "none",
            "releaseNameFormat": null,
            "keepForever": false,
            "definitionSnapshotRevision": 0,
            "logsContainerUrl": null,
            "_links": {},
            "tags": [],
            "triggeringArtifactAlias": null,
            "projectReference": null
        },
        "project": {
            "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-
0000-000000000000",
```

```
            "name": "Fabrikam"
        }
    },
    "resourceVersion": "3.0-preview.1",
    "resourceContainers": {
        "collection": {
            "id": "00000000-0000-0000-0000-000000000000"
        },
        "account": {
            "id": "00000000-0000-0000-0000-000000000000"
        },
        "project": {
            "id": "00000000-0000-0000-0000-000000000000"
        }
    },
    "createdDate": "2019-10-10T17:49:39.157Z"
}
```


Pipeline

Check updated

Event: A check is updated.

- Publisher ID: `azure-devops`
- Event ID: `check.updated`
- Resource Name: `check`

Settings

- `project`: Filter events to include only checks updated within the specified project.
- `checkType`: Filter events to include only checks of the specified type.
- `status`: Filter events to include only checks with the specified status.

Sample payload

JSON


```
{
  "publisherId": "azure-devops",
  "eventId": "check.updated",
  "resource": {
    "check": {
      "id": "00000000-0000-0000-0000-000000000000check-id",
      "type": "check-type",
```

```
      "status": "check-status",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000project-id",
        "name": "project-name"
      }
    }
  },
  "updatedDate": "2024-07-17T21:34:22.338Z"
}
```


Elastic agent pool resized

Event: An elastic agent pool is resized.

- Publisher ID: `azure-devops`
- Event ID: `elasticagentpool.resized`
- Resource Name: `elasticagentpool`

Settings

- `project`: Filter events to include only elastic agent pools resized within the specified project.
- `poolName`: Filter events to include only elastic agent pools with the specified name pattern.
- `oldSize`: Filter events to include only elastic agent pools with the specified old size.
- `newSize`: Filter events to include only elastic agent pools with the specified new size.

Sample payload

JSON


```
{
  "publisherId": "azure-devops",
  "eventId": "elasticagentpool.resized",
  "resource": {
    "elasticAgentPool": {
      "id": "00000000-0000-0000-0000-000000000000pool-id",
      "name": "pool-name",
      "oldSize": "old-size",
      "newSize": "new-size",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000project-id",
        "name": "project-name"
      }
```

```
    }
  },
  "resizedDate": "2024-07-17T21:34:22.338Z"
}
```


Manual intervention pending

Event: A manual intervention is pending.

- Publisher ID: `azure-devops`
- Event ID: `manualintervention.pending`
- Resource Name: `manualintervention`

Settings

- `project`: Filter events to include only manual interventions pending within the specified project.
- `interventionName`: Filter events to include only manual interventions with the specified name pattern.
- `status`: Filter events to include only manual interventions with the specified status.

Sample payload

JSON


```
{
  "publisherId": "azure-devops",
  "eventId": "manualintervention.pending",
  "resource": {
    "manualIntervention": {
      "id": "00000000-0000-0000-0000-000000000000intervention-id",
      "name": "intervention-name",
      "status": "pending",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000project-id",
        "name": "project-name"
      }
    }
  },
  "createdDate": "2024-07-17T21:34:22.338Z"
}
```


Project-level agent pool created

Event: A project-level agent pool is created.

- Publisher ID: `azure-devops`
- Event ID: `projectlevelagentpool.created`
- Resource Name: `projectlevelagentpool`

Settings

- `project`: Filter events to include only project-level agent pools created within the specified project.
- `poolName`: Filter events to include only project-level agent pools with the specified name pattern.

Sample payload

JSON


```
{
  "publisherId": "azure-devops",
  "eventId": "projectlevelagentpool.created",
  "resource": {
    "projectLevelAgentPool": {
      "id": "00000000-0000-0000-0000-000000000000pool-id",
      "name": "pool-name",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000project-id",
        "name": "project-name"
      }
    }
  },
  "createdDate": "2024-07-17T21:34:22.338Z"
}
```


Project-level agent pool updated

Event: A project-level agent pool is updated.

- Publisher ID: `azure-devops`
- Event ID: `projectlevelagentpool.updated`
- Resource Name: `projectlevelagentpool`

Settings

- `project`: Filter events to include only project-level agent pools updated within the specified project.
- `poolName`: Filter events to include only project-level agent pools with the specified name pattern.

Sample payload

JSON


```
{
    "publisherId": "azure-devops",
    "eventId": "projectlevelagentpool.updated",
    "resource": {
        "projectLevelAgentPool": {
            "id": "00000000-0000-0000-0000-000000000000pool-id",
            "name": "pool-name",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            }
        }
    },
    "updatedDate": "2024-07-17T21:34:22.338Z"
}
```


Run state changed

Event: Overall statuses of a pipeline run changed. A new run started, or a run transitioned to canceling, canceled, failed, partially succeeded or succeeded state.

- Publisher ID: `pipelines`
- Event ID: `ms.vss-pipelines.run-state-changed-event`
- Resource Name: `resource`

Settings

- `PipelineId`: Filter to include only events for the specified pipeline
- `runStateId`: Filter events based on the new state of the run
  - Valid values:
    - `InProgress`
    - `Canceling`
    - `Completed`

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "ms.vss-pipelines.run-state-changed-event",
  "publisherId": "pipelines",
  "message": {
    "text": "Run 11 succeeded.",
    "html": "Run <a href=\"https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/results?buildId=11\">11</a> succeeded.",
    "markdown": "Run [11](https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/results?buildId=11) succeeded."
  },
  "detailedMessage": {
    "text": "Run 11 succeeded.",
    "html": "Run <a href=\"https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/results?buildId=11\">11</a> succeeded.",
    "markdown": "Run [11](https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/results?buildId=11) succeeded."
  },
  "resource": {
    "run": {
      "_links": {
        "self": {
          "href": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_apis/Pipelines/1/runs/11"
        },
        "web": {
          "href": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/results?buildId=11"
        }
      },
      "pipeline": {
        "id": 1,
        "name": "Pipeline-Name"
      },
      "state": "completed",
      "result": "succeeded",
      "createdDate": "2024-07-17T21:34:22.338Z",
      "finishedDate": "2024-07-17T21:45:22.338Z",
      "url": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_apis/Pipelines/1/runs/11"
    }
  },
  "createdDate": "2024-07-17T21:34:22.338Z"
}
```


Run stage state changed

Event: A new stage started, or a stage transitioned to canceling, canceled, failed, partially succeeded, or succeeded.

- Publisher ID: `pipelines`
- Event ID: `ms.vss-pipelines.stage-state-changed-event`
- Resource Name: `resource`

Settings

- `PipelineId`: Filter to include only events for the specified pipeline
- `stageNameId`: Filter events to a specific stage name
- `stageStateId`: Filter events based on the new state of the stage
  - Valid values:
    - `NotStarted`
    - `Waiting`
    - `Running`
    - `Completed`

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000ac1dd6da-af30-43cb-8434-
e1005864b0a3",
  "eventType": "ms.vss-pipelines.stage-state-changed-event",
  "publisherId": "pipelines",
  "message": {
    "text": "Run 2 stage __default succeeded.",
    "html": "Run 2 stage <a href=\"https://codedev.ms/org/00000000-0000-
0000-0000-000000000000/_build/results?buildId=2\">__default</a> succeeded.",
    "markdown": "Run 2 stage [__default](https://codedev.ms/org/00000000-
0000-0000-0000-000000000000/_build/results?buildId=2) succeeded."
  },
  "detailedMessage": {
    "text": "Run 2 stage __default succeeded.",
    "html": "Run 2 stage <a href=\"https://codedev.ms/org/00000000-0000-
0000-0000-000000000000/_build/results?buildId=2\">__default</a> succeeded.",
    "markdown": "Run 2 stage [__default](https://codedev.ms/org/00000000-
0000-0000-0000-000000000000/_build/results?buildId=2) succeeded."
  },
  "resource": {
    "stage": {
      "_links": {
        "web": {
          "href": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/results?buildId=2"
```

```
        },
        "pipeline.web": {
          "href": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_build/definition?definitionId=2"
        }
      },
      "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-0000-
000000000000",
      "name": "__default",
      "displayName": null,
      "state": "completed",
      "result": "succeeded"
    },
    "run": {
      "pipeline": {
        "url": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_apis/Pipelines/2?revision=2",
        "id": 2,
        "revision": 2,
        "name": "TEST-CI",
        "folder": "\\"
      },
      "state": "completed",
      "result": "succeeded",
      "createdDate": "2019-12-13T06:10:10.164Z",
      "finishedDate": "2019-12-13T06:10:10.164Z",
      "id": 2,
      "name": "2"
    },
    "pipeline": {
      "url": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_apis/Pipelines/2?revision=2",
      "id": 2,
      "revision": 2,
      "name": "TEST-CI",
      "folder": "\\"
    },
    "repositories": [
      {
        "type": "Git",
        "change":
        {
          "author":
          {
            "name": "Fabrikam John",
            "email": "john@fabrikamfiber.com",
            "date": "2024-11-11T15:09:21Z"
          },
          "committer":
          {
            "name": "Fabrikam John",
            "email": "john@fabrikamfiber.com",
            "date": "2024-11-11T15:09:21Z"
          },
          "message": "Added Viva support"
```

```
        },
        "url":
"https://fabrikamfiber@dev.azure.com/fabrikamfiber/fabrikamfiber-
viva/_git/fabrikamfiber"
      }
    ]
  },
  "resourceVersion": "5.1-preview.1",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2019-12-13T06:10:10.186Z"
}
```


Run stage waiting for approval

Event: An approval is created for a run stage.

- Publisher ID: `pipelines`
- Event ID: `ms.vss-pipelinechecks-events.approval-pending`
- Resource Name: `resource`

Settings

- `PipelineId`: Filter to include only events for the specified pipeline
- `stageName`: Filter events to a specific stage name
- `environmentName`: Filter events to approvals for deployments to a specified environment

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000055382df7-24fa-453c-9173-
3369b2417a5b",
  "eventType": "ms.vss-pipelinechecks-events.approval-pending",
  "publisherId": "pipelines",
  "message": {
```

```
    "text": "Approval pending for deployment of pipeline run1 to environment
env1.",
    "html": "Approval pending for deployment of pipeline <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results> run1 </a> to environment <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources>env1</a>.",
    "markdown": "Approval pending for deployment of pipeline
[https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results](run1) to environment [env1]
(https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources)"
  },
  "detailedMessage": {
    "text": "Approval pending for deployment of pipeline run1 to environment
env1.",
    "html": "Approval pending for deployment of pipeline <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results> run1 </a> to environment <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources>env1</a>.",
    "markdown": "Approval pending for deployment of pipeline
[https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results](run1) to environment [env1]
(https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources)"
  },
  "resource": {
    "approval": {
      "id": "00000000-0000-0000-0000-0000000000000f027b05-0942-4a35-9218-
26fa07d8760a",
      "steps": [
        {
          "assignedApprover": {
            "displayName": null,
            "id": "00000000-0000-0000-0000-000000000000743f73b7-cdeb-4de7-
80b7-00cee17476b8"
          },
          "status": "pending",
          "comment": "Sample comment",
          "initiatedOn": "2019-12-13T06:14:11.642Z"
        }
      ],
      "status": "pending",
      "createdOn": "2019-12-13T06:14:11.642Z",
      "lastModifiedOn": "2019-12-13T06:14:11.642Z",
      "instructions": "Instructions",
      "minRequiredApprovers": 2,
      "blockedApprovers": [
        {
          "displayName": null,
          "id": "00000000-0000-0000-0000-000000000000d651e716-a205-4b37-
a803-e373df09fea6"
        }
      ],
```

```
      "_links": {}
    },
    "projectId": "00000000-0000-0000-0000-000000000000",
    "pipeline": null,
    "stage": null,
    "run": null,
    "resource": null,
    "id": 0,
    "url": null,
    "stageName": null,
    "attemptId": 0
  },
  "resourceVersion": "5.1-preview.1",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2019-12-13T06:14:11.671Z"
}
```


Run stage approval completed

Event: An approval completed for a run stage.

- Publisher ID: `pipelines`
- Event ID: `ms.vss-pipelinechecks-events.approval-completed`
- Resource Name: `resource`

Settings

- `PipelineId`: Filter to include only events for the specified pipeline
- `stageName`: Filter events to a specific stage name
- `environmentName`: Filter events to approvals for deployments to a specified environment

Sample payload

JSON

```
{
  "id": "00000000-0000-0000-0000-000000000005810cce3-55e9-46dc-ad4f-
681c57cf620e",
  "eventType": "ms.vss-pipelinechecks-events.approval-completed",
  "publisherId": "pipelines",
  "message": {
    "text": "Approval completed for deployment of pipeline run1 to
environment env1.",
    "html": "Approval completed for deployment of pipeline <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results> run1 </a> to environment <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources>env1</a>.",
    "markdown": "Approval completed for deployment of pipeline
[https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results](run1) to environment [env1]
(https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources)"
  },
  "detailedMessage": {
    "text": "Approval completed for deployment of pipeline run1 to
environment env1.",
    "html": "Approval completed for deployment of pipeline <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results> run1 </a> to environment <a
href=https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources>env1</a>.",
    "markdown": "Approval completed for deployment of pipeline
[https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_build/results?
buildId=2&view=results](run1) to environment [env1]
(https://dev.azure.com/fabfiber/Fabrikam-Fiber-Git/_environments/1?
view=resources)"
  },
  "resource": {
    "approval": {
      "id": "00000000-0000-0000-0000-0000000000000f027b05-0942-4a35-9218-
26fa07d8760a",
      "steps": [
        {
          "assignedApprover": {
            "displayName": null,
            "id": "00000000-0000-0000-0000-000000000000f8482ec0-3e2f-489b-
ba62-ea01cf84afa8"
          },
          "status": "approved",
          "comment": "Sample comment",
          "initiatedOn": "2019-12-13T06:18:22.460Z"
        }
      ],
      "status": "approved",
      "createdOn": "2019-12-13T06:18:22.460Z",
      "lastModifiedOn": "2019-12-13T06:18:22.460Z",
      "instructions": "Instructions",
      "minRequiredApprovers": 2,
```

```
      "blockedApprovers": [
        {
          "displayName": null,
          "id": "00000000-0000-0000-0000-0000000000023241e2e-59af-4b58-
842e-5604d508c6b5"
        }
      ],
      "_links": {}
    },
    "projectId": "00000000-0000-0000-0000-000000000000",
    "pipeline": null,
    "stage": null,
    "run": null,
    "resource": null,
    "id": 0,
    "url": null,
    "stageName": null,
    "attemptId": 0
  },
  "resourceVersion": "5.1-preview.1",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2019-12-13T06:18:22.487Z"
}
```


Run job state changed

Event: A new job is running, or it completed, or is waiting for an agent.

- Publisher ID: `pipelines`
- Event ID: `ms.vss-pipelines.job-state-changed-event`
- Resource Name: `resource`

Settings

- `pipelineId`: Filter to include only events for the specified pipeline
- `stageNameId`: Filter events to a specific stage name
- `jobNameId`: Filter events to a specific job name
- `jobStateId`: Filter events based on the new state of the job

    - Valid values:
        - `Waiting`
        - `Running`
        - `Completed`
- `jobResultId`: Filter events based on the result of the job
    - Valid values:
        - `Succeeded`
        - `Skipped`
        - `Rejected`
        - `Failed`
        - `Canceled`

Sample payload

JSON


```
{
    "subscriptionId": "00000000-0000-0000-0000-000000000000",
    "notificationId": 3,
    "id": "00000000-0000-0000-0000-000000000000ac1dd6da-af30-43cb-8434-
e1005864b0a3",
    "eventType": "ms.vss-pipelines.job-state-changed-event",
    "publisherId": "pipelines",
    "message":
    {
        "text": "Run 20241121.5 stage Build job Compile succeeded.",
        "html": "Run 20241121.5 stage Build job <a
href=\"https://dev.azure.com/fabrikamfiber/fabrikamfiber-
viva/_build/results?buildId=2710088\">Compile</a> succeeded.",
        "markdown": "Run 20241121.5 stage Build job [Compile]
(https://dev.azure.com/fabrikamfiber/fabrikamfiber-viva/_build/results?
buildId=2710088) succeeded."
    },
    "detailedMessage":
    {
        "text": "Run 20241121.5 stage Build job Compile succeeded.",
        "html": "Run 20241121.5 stage Build job <a
href=\"https://dev.azure.com/fabrikamfiber/fabrikamfiber-
viva/_build/results?buildId=2710088\">Compile</a> succeeded.",
        "markdown": "Run 20241121.5 stage Build job [Compile]
(https://dev.azure.com/fabrikamfiber/fabrikamfiber-viva/_build/results?
buildId=2710088) succeeded."
    },
    "resource":
    {
        "job":
        {
            "_links":
```

```
            {
                "web":
                {
                    "href":
"https://dev.azure.com/fabrikamfiber/fabrikamfiber-viva/_build/results?
buildId=2"
                },
                "pipeline.web":
                {
                    "href":
"https://dev.azure.com/fabrikamfiber/fabrikamfiber-viva/_build/definition?
definitionId=2"
                }
            },
            "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-
0000-000000000000",
            "name": "__default",
            "state": "completed",
            "result": "succeeded",
            "startTime": "2024-11-21T16:42:52.7761408Z",
            "finishTime": "2024-11-21T16:42:52.7761408Z"
        },
        "stage":
        {
            "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-
0000-000000000000",
            "name": "__default",
            "displayName": null,
            "state": "completed",
            "result": "succeeded",
            "startTime": null,
            "finishTime": null
        },
        "run":
        {
            "pipeline":
            {
                "url": "https://codedev.ms/org/1234/_apis/Pipelines/2?
revision=2",
                "id": 2,
                "revision": 2,
                "name": "TEST-CI",
                "folder": "\\"
            },
            "state": "completed",
            "result": "succeeded",
            "createdDate": "2024-11-21T16:42:52.7761408Z",
            "finishedDate": "2024-11-21T16:42:52.7761408Z",
            "id": 2,
            "name": "2"
        },
        "pipeline":
        {
            "url": "https://codedev.ms/org/00000000-0000-0000-0000-
000000000000/_apis/Pipelines/2?revision=2",
```

```
            "id": 2,
            "revision": 2,
            "name": "TEST-CI",
            "folder": "\\"
        },
        "repositories":
        [
            {
                "type": "Git",
                "change":
                {
                    "author":
                    {
                        "name": "Fabrikam John",
                        "email": "john@fabrikamfiber.com",
                        "date": "2024-11-11T15:09:21Z"
                    },
                    "committer":
                    {
                        "name": "Fabrikam John",
                        "email": "john@fabrikamfiber.com",
                        "date": "2024-11-11T15:09:21Z"
                    },
                    "message": "Added Viva support"
                },
                "url":
"https://fabrikamfiber@dev.azure.com/fabrikamfiber/fabrikamfiber-
viva/_git/fabrikamfiber"
            },
            {
                "type": "GitHub",
                "change":
                {
                    "author":
                    {
                        "name": "Fabrikam John",
                        "email": "john@github.com",
                        "date": "2024-08-11T15:05:20Z"
                    },
                    "committer":
                    {
                        "name": "Fabrikam John",
                        "email": "john@github.com",
                        "date": "2024-08-11T15:05:20Z"
                    },
                    "message": "Added Viva open source REST API library"
                },
                "url": "https://api.github.com/repos/FabrikamFiber/Viva"
            }
        ]
    },
    "resourceVersion": "5.1-preview.1",
    "resourceContainers":
    {
        "collection":
```

```
        {
            "id": "00000000-0000-0000-0000-000000000000"
        },
        "account":
        {
            "id": "00000000-0000-0000-0000-000000000000"
        },
        "project":
        {
            "id": "00000000-0000-0000-0000-000000000000"
        }
    },
    "createdDate": "2024-11-21T16:42:53.5254422Z"
}
```


Code

Code checked in

Event: A changeset is checked into Team Foundation Version Control (TFVC).

- Publisher ID: `azure-devops`
- Event ID: `tfvc.checkin`
- Resource Name: `changeset`

Settings

- `path`: Filter to checkins that change one or more files under the specified path
  - Required

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "tfvc.checkin",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Normal Paulk checked in changeset 18: Dropping in new Java
sample",
    "html": "Normal Paulk checked in changeset <a
href=\"https://dev.azure.com/fabrikam-fiber-inc/web/cs.aspx?pcguid=000-0000-
0000\">18</a>: Dropping in new Java sample",
    "markdown": "Chuck Reinhart checked in changeset [18]
```

```
(https://dev.azure.com/fabrikam-fiber-inc/web/cs.aspx?pcguid=000-0000-0000):
Dropping in new Java sample"
  },
  "detailedMessage": {
    "text": "Chuck Reinhart checked in changeset 18: Dropping in new Java
sample",
    "html": "Chuck Reinhart checked in changeset <a
href=\"https://dev.azure.com/fabrikam-fiber-inc/web/cs.aspx?pcguid=d81542e4-
cdfa-4333-b082-1ae2d6c3ad16&amp;cs=18\">18</a>: Dropping in new Java
sample",
    "markdown": "Chuck Reinhart checked in changeset [18]
(https://dev.azure.com/fabrikam-fiber-inc/web/cs.aspx?pcguid=000-0000-0000):
Dropping in new Java sample"
  },
  "resource": {
    "changesetId": 18,
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/tfvc/changesets/18",
    "author": {
      "id": "00000000-0000-0000-0000-000000000000",
      "displayName": "Chuck Reinhart",
      "uniqueName": "fabrikamfiber16@hotmail.com"
    },
    "checkedInBy": {
      "id": "00000000-0000-0000-0000-000000000000",
      "displayName": "Chuck Reinhart",
      "uniqueName": "fabrikamfiber16@hotmail.com"
    },
    "createdDate": "2014-05-12T22:41:16Z",
    "comment": "Dropping in new Java sample"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:26.2056408Z"
}
```


Code pushed

Event: Code is pushed to a Git repository.

- Publisher ID: `azure-devops`
- Event ID: `git.push`
- Resource Name: `push`

Settings

- `branch`: The branch that code is pushed into
- `pushedBy`: A group that has the pusher as its member
- `repository`: The repository that code is pushed to
  - Data type: `guid`

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-0000000000000",
  "eventType": "git.push",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Jamal Hartnett pushed updates to branch master of repository
Fabrikam-Fiber-Git.",
    "html": "Jamal Hartnett pushed updates to branch master of repository
Fabrikam-Fiber-Git.",
    "markdown": "Jamal Hartnett pushed updates to branch `master` of
repository `Fabrikam-Fiber-Git`."
  },
  "detailedMessage": {
    "text": "Jamal Hartnett pushed 1 commit to branch master of repository
Fabrikam-Fiber-Git.\n - Fixed bug in web.config file 33b55f7c",
    "html": "Jamal Hartnett pushed 1 commit to branch <a
href=\"https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_git/Fabrikam-Fiber-Git/#version=GBmaster\">master</a>
of repository <a href=\"https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_git/Fabrikam-Fiber-Git/\">Fabrikam-Fiber-
Git</a>.\n<ul>\n- Fixed bug in web.config file <a
href=\"https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_git/Fabrikam-Fiber-Git/commit/00000000-0000-0000-
0000-000000000000\">33b55f7c</a>\n</ul>",
    "markdown": "Jamal Hartnett pushed 1 commit to branch [master]
(https://dev.azure.com/fabrikam-fiber-inc/DefaultCollection/_git/Fabrikam-
Fiber-Git/#version=GBmaster) of repository [Fabrikam-Fiber-Git]
(https://dev.azure.com/fabrikam-fiber-inc/DefaultCollection/_git/Fabrikam-
Fiber-Git/).\n* Fixed bug in web.config file [33b55f7c]
(https://dev.azure.com/fabrikam-fiber-inc/DefaultCollection/_git/Fabrikam-
Fiber-Git/commit/00000000-0000-0000-0000-000000000000)"
  },
  "resource": {
    "commits": [
      {
        "commitId": "00000000-0000-0000-0000-000000000000",
        "author": {
          "name": "Jamal Hartnett",
          "email": "fabrikamfiber4@hotmail.com",
```

```
          "date": "2024-02-25T19:01:00Z"
        },
        "committer": {
          "name": "Jamal Hartnett",
          "email": "fabrikamfiber4@hotmail.com",
          "date": "2024-02-25T19:01:00Z"
        },
        "comment": "Fixed bug in web.config file",
        "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_git/Fabrikam-Fiber-Git/commit/00000000-0000-0000-
0000-000000000000"
      }
    ],
    "refUpdates": [
      {
        "name": "refs/heads/main",
        "oldObjectId": "aad331d8d3b131fa9ae03cf5e53965b51942618a",
        "newObjectId": "00000000-0000-0000-0000-000000000000"
      }
    ],
    "repository": {
      "id": "00000000-0000-0000-0000-000000000000",
      "name": "Fabrikam-Fiber-Git",
      "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/repos/git/repositories/abcd-1234-efgh-5678",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000",
        "name": "Fabrikam-Fiber-Git",
        "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/projects/abcd-1234-efgh-5678",
        "state": "wellFormed"
      },
      "defaultBranch": "refs/heads/main",
      "remoteUrl": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_git/Fabrikam-Fiber-Git"
    },
    "pushedBy": {
      "id": "00000000-0000-0000-0000-000000000000@Live.com",
      "displayName": "Jamal Hartnett",
      "uniqueName": "Windows Live ID\\fabrikamfiber4@hotmail.com"
    },
    "pushId": 14,
    "date": "2014-05-02T19:17:13.3309587Z",
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/repos/git/repositories/abcd-1234-efgh-
5678/pushes/14"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
```

```
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:27.0379153Z"
}
```


Pull request created

Event: A pull request is created in a Git repository.

- Publisher ID: `azure-devops`
- Event ID: `git.pullrequest.created`
- Resource Name: `pullrequest`

Settings

- `repository`: The repository that code is pushed to
  - Data type: `guid`
- `pullrequestCreatedBy`: A group that has the requester as a member
- `pullrequestReviewersContains`: A group included in the reviewers list
- `branch`: The target branch of the pull request

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "git.pullrequest.created",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Jamal Hartnett created a new pull request",
    "html": "Jamal Hartnett created a new pull request",
    "markdown": "Jamal Hartnett created a new pull request"
  },
  "detailedMessage": {
    "text": "Jamal Hartnett created a new pull request\r\n\r\n- Merge
status: Succeeded\r\n- Merge commit:
eef717(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repo
sitories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-
0000-000000000000)\r\n",
    "html": "Jamal Hartnett created a new pull request\r\n<ul>\r\n- Merge
status: Succeeded</br>\r\n- Merge commit: <a
href=\"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repo
```

```
sitories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-
0000-000000000000\">eef717</a></br>\r\n</ul>",
    "markdown": "Jamal Hartnett created a new pull request\r\n\r\n+ Merge
status: Succeeded\r\n+ Merge commit: [eef717]
(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000)\r\n"
  },
  "resource": {
    "repository": {
      "id": "00000000-0000-0000-0000-000000000000",
      "name": "Fabrikam",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000",
        "name": "Fabrikam",
        "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/projects/abcd-1234-
efgh-5678",
        "state": "wellFormed"
      },
      "defaultBranch": "refs/heads/main",
      "remoteUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_git/Fabrikam"
    },
    "pullRequestId": 1,
    "status": "active",
    "createdBy": {
      "id": "00000000-0000-0000-0000-000000000000",
      "displayName": "Jamal Hartnett",
      "uniqueName": "fabrikamfiber4@hotmail.com",
      "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-0000-
0000-0000-000000000000",
      "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000"
    },
    "creationDate": "2024-06-17T11:22:33.456789Z",
    "title": "my first pull request",
    "description": " - test2\r\n",
    "sourceRefName": "refs/heads/mytopic",
    "targetRefName": "refs/heads/main",
    "mergeStatus": "succeeded",
    "mergeId": "a1234567-abcde-1234-abc",
    "lastMergeSourceCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "lastMergeTargetCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
```

```
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "lastMergeCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "reviewers": [
      {
        "reviewerUrl": null,
        "vote": 0,
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "[Mobile]\\Mobile Team",
        "uniqueName": "azure-devops:///Classification/TeamProject/00000000-
0000-0000-0000-000000000000\\Mobile Team",
        "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-
0000-0000-0000-000000000000",
        "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000",
        "isContainer": true
      }
    ],
    "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/pullRequests/1"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:27.2879096Z"
}
```


Pull request merge attempted

Event: A pull request merge is attempted in a Git repository.

- Publisher ID: `azure-devops`

- Event ID: `git.pullrequest.merge.attempted`
- Resource Name: `pullrequest`

Settings

- `repository`: The repository that code is pushed to
  - Data type: `guid`
- `pullrequestCreatedBy`: A group that has the requester as a member
- `pullrequestReviewersContains`: A group included in the reviewers list
- `branch`: The target branch of the pull request

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "git.pullrequest.merge.attempted",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Jamal Hartnett attempted to merge a pull request",
    "html": "Jamal Hartnett attempted to merge a pull request",
    "markdown": "Jamal Hartnett attempted to merge a pull request"
  },
  "detailedMessage": {
    "text": "Jamal Hartnett attempted to merge a pull request\r\n\r\n- Merge
status: Attempted\r\n- Merge commit:
eef717(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repo
sitories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-
0000-000000000000)\r\n",
    "html": "Jamal Hartnett created a new pull request\r\n<ul>\r\n- Merge
status: Attempted</br>\r\n- Merge commit: <a
href=\"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repo
sitories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-
0000-000000000000\">eef717</a></br>\r\n</ul>",
    "markdown": "Jamal Hartnett created a new pull request\r\n\r\n+ Merge
status: Attempted\r\n+ Merge commit: [eef717]
(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000)\r\n"
  },
  "resource": {
    "repository": {
      "id": "00000000-0000-0000-0000-000000000000",
      "name": "Fabrikam",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000",
```

```
      "project": {
        "id": "00000000-0000-0000-0000-000000000000abcd-1234-efgh-5678",
        "name": "Fabrikam",
        "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/projects/00000000-
0000-0000-0000-000000000000",
        "state": "wellFormed"
      },
      "defaultBranch": "refs/heads/main",
      "remoteUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_git/Fabrikam"
    },
    "pullRequestId": 1,
    "status": "active",
    "createdBy": {
      "id": "00000000-0000-0000-0000-000000000000",
      "displayName": "Jamal Hartnett",
      "uniqueName": "fabrikamfiber4@hotmail.com",
      "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-0000-
0000-0000-000000000000",
      "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000"
    },
    "creationDate": "2024-06-17T11:22:33.456789Z",
    "title": "my first pull request",
    "description": " - test2\r\n",
    "sourceRefName": "refs/heads/mytopic",
    "targetRefName": "refs/heads/main",
    "mergeStatus": "attempted",
    "mergeId": "00000000-0000-0000-0000-000000000000",
    "lastMergeSourceCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "lastMergeTargetCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "lastMergeCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "reviewers": [
      {
        "reviewerUrl": null,
```

```
        "vote": 0,
        "id": "00000000-0000-0000-0000-000000000000",
        "displayName": "[Mobile]\\Mobile Team",
        "uniqueName": "azure-devops:///Classification/TeamProject/00000000-
0000-0000-0000-000000000000\\Mobile Team",
        "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-
0000-0000-0000-000000000000",
        "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000",
        "isContainer": true
      }
    ],
    "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/pullRequests/1"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:27.2879096Z"
}
```


Pull request merge approved

Event: A merge commit is approved on a pull request.

- Publisher ID: `azure-devops`
- Event ID: `git.pullrequest.approved`
- Resource Name: `pullrequest`

Settings

- `repository`: The repository that code is pushed to
  - Data type: `guid`
- `pullrequestCreatedBy`: A group that has the requester as a member
- `pullrequestReviewersContains`: A group included in the reviewers list
- `branch`: The target branch of the pull request

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "git.pullrequest.merged",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Jamal Hartnett has created a pull request merge approved",
    "html": "Jamal Hartnett has created a pull request merge approved",
    "markdown": "Jamal Hartnett has created a pull request merge approved"
  },
  "detailedMessage": {
    "text": "Jamal Hartnett has created a pull request merge
approved\r\n\r\n- Merge status: Succeeded\r\n- Merge commit:
eef717(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repo
sitories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-
0000-000000000000)\r\n",
    "html": "Jamal Hartnett has created a pull request merge
approved\r\n<ul>\r\n- Merge status: Succeeded</br>\r\n- Merge commit: <a
href=\"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repo
sitories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-
0000-000000000000\">eef717</a></br>\r\n</ul>",
    "markdown": "Jamal Hartnett has created a pull request merge
approved\r\n\r\n+ Merge status: Succeeded\r\n+ Merge commit: [eef717]
(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000)\r\n"
  },
  "resource": {
    "repository": {
      "id": "00000000-0000-0000-0000-000000000000000000000-0000-0000-0000-
000000000000",
      "name": "Fabrikam",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000abcd-1234-efgh-5678",
        "name": "Fabrikam",
        "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/projects/abcd-1234-
efgh-5678",
        "state": "wellFormed"
      },
      "defaultBranch": "refs/heads/main",
      "remoteUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_git/Fabrikam"
    },
    "pullRequestId": 1,
    "status": "completed",
    "createdBy": {
```

```
      "id": "00000000-0000-0000-0000-000000000000000000-0000-0000-0000-
000000000000",
      "displayName": "Jamal Hartnett",
      "uniqueName": "fabrikamfiber4@hotmail.com",
      "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-0000-
0000-0000-000000000000",
      "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000"
    },
    "creationDate": "2024-06-17T11:22:33.456789Z",
    "closedDate": "2024-06-30T18:59:12.3660573Z",
    "title": "my first pull request",
    "description": " - test2\r\n",
    "sourceRefName": "refs/heads/mytopic",
    "targetRefName": "refs/heads/main",
    "mergeStatus": "succeeded",
    "mergeId": "a1234567-abcde-1234-abc",
    "lastMergeSourceCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "lastMergeTargetCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "lastMergeCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
    },
    "reviewers": [
      {
        "reviewerUrl": null,
        "vote": 0,
        "id": "00000000-0000-0000-0000-000000000000000000-0000-0000-0000-
000000000000",
        "displayName": "[Mobile]\\Mobile Team",
        "uniqueName": "azure-devops:///Classification/TeamProject/00000000-
0000-0000-0000-000000000000\\Mobile Team",
        "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-
0000-0000-0000-000000000000",
        "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000",
        "isContainer": true
      }
```

```
    ],
    "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/pullRequests/1"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:27.3156388Z"
}
```


Pull request updated

Event: A pull request is updated; status, review list, reviewer vote changed, or the source branch is updated with a push.

- Publisher ID: `azure-devops`
- Event ID: `git.pullrequest.updated`
- Resource Name: `pullrequest`

Settings

- `notificationType`: The type of pull request change
  - Valid values:
    - `PushNotification` - Source branch updated
    - `ReviewersUpdateNotification` - Reviewers changed
    - `StatusUpdateNotification` - Status changed
    - `ReviewerVoteNotification` - Votes score changed
- `repository`: The repository that code is pushed to
  - Data type: `guid`
- `pullrequestCreatedBy`: A group that has the requester as a member
- `pullrequestReviewersContains`: A group included in the reviewers list
- `branch`: The target branch of the pull request

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "git.pullrequest.updated",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Jamal Hartnett marked the pull request as completed",
    "html": "Jamal Hartnett marked the pull request as completed",
    "markdown": "Jamal Hartnett marked the pull request as completed"
  },
  "detailedMessage": {
    "text": "Jamal Hartnett marked the pull request as completed\r\n\r\n- Merge status: Succeeded\r\n- Merge commit: eef717(https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-000000000000)\r\n",
    "html": "Jamal Hartnett marked the pull request as completed\r\n<ul>\r\n- Merge status: Succeeded</br>\r\n- Merge commit: <a href=\"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-000000000000\">eef717</a></br>\r\n</ul>",
    "markdown": "Jamal Hartnett marked the pull request as completed\r\n\r\n+ Merge status: Succeeded\r\n+ Merge commit: [eef717](https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositories/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-000000000000)\r\n"
  },
  "resource": {
    "repository": {
      "id": "00000000-0000-0000-0000-000000000000",
      "name": "Fabrikam",
      "url": "https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositories/00000000-0000-0000-0000-000000000000",
      "project": {
        "id": "00000000-0000-0000-0000-000000000000",
        "name": "Fabrikam",
        "url": "https://dev.azure.com/fabrikam/DefaultCollection/_apis/projects/00000000-0000-0000-0000-000000000000",
        "state": "wellFormed"
      },
      "defaultBranch": "refs/heads/main",
      "remoteUrl": "https://dev.azure.com/fabrikam/DefaultCollection/_git/Fabrikam"
    },
    "pullRequestId": 1,
    "status": "completed",
    "createdBy": {
      "id": "00000000-0000-0000-0000-000000000000000000-0000-0000-0000-
```

```
000000000000",
      "displayName": "Jamal Hartnett",
      "uniqueName": "fabrikamfiber4@hotmail.com",
      "url": "https://dev.azure.com/fabrikam/_apis/Identities/00000000-0000-
0000-0000-000000000000",
      "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=00000000-0000-0000-0000-000000000000"
    },
    "creationDate": "2024-06-17T16:55:46.589889Z",
    "closedDate": "2024-06-30T18:59:12.3660573Z",
    "title": "my first pull request",
    "description": " - test2\r\n",
    "sourceRefName": "refs/heads/mytopic",
    "targetRefName": "refs/heads/main",
    "mergeStatus": "succeeded",
    "mergeId": "00000000-0000-0000-0000-000000000000",
    "lastMergeSourceCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/abc-defg-123-4567/commits/00000000-0000-0000-0000-000000000000"
    },
    "lastMergeTargetCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/abc-defg-123-4567/commits/00000000-0000-0000-0000-000000000000"
    },
    "lastMergeCommit": {
      "commitId": "00000000-0000-0000-0000-000000000000",
      "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/abc-defg-123-4567/commits/00000000-0000-0000-0000-000000000000"
    },
    "reviewers": [
      {
        "reviewerUrl": null,
        "vote": 0,
        "id": "00000000-0000-0000-0000-000000000000000000-0000-0000-0000-
000000000000",
        "displayName": "[Mobile]\\Mobile Team",
        "uniqueName": "azure-devops:///Classification/TeamProject/00000000-
0000-0000-0000-000000000000\\Mobile Team",
        "url":
"https://dev.azure.com/fabrikam/_apis/Identities/abcdefg1234567",
        "imageUrl":
"https://dev.azure.com/fabrikam/DefaultCollection/_api/_common/identityImage
?id=abcdefg1234567",
        "isContainer": true
      }
    ],
    "commits": [
      {
        "commitId": "00000000-0000-0000-0000-000000000000",
```

```
        "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
      }
    ],
    "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/repos/git/repositori
es/00000000-0000-0000-0000-000000000000/pullRequests/1"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:27.2813828Z"
}
```


Pull request commented on

Event: A pull request is commented on.

- Publisher ID: `azure-devops`
- Event ID: `git-pullrequest-comment-event`
- Resource Name: `pullrequest`

Settings

- `repository`: The repository that pull request is commented on
  - Data type: `guid`
- `branch`: The target branch of the pull request

Sample payload

JSON


```
{
    "id": "00000000-0000-0000-0000-000000000000af07be1b-f3ad-44c8-a7f1-
c4835f2df06b",
    "eventType": "ms.vss-code.git-pullrequest-comment-event",
```

```
    "publisherId": "azure-devops",
    "message": {
        "text": "Jamal Hartnett has edited a pull request comment",
        "html": "Jamal Hartnett has <a
href=\"https://fabrikam.visualstudio.com/DefaultCollection/_git/Fabrikam/pul
lrequest/1?discussionId=5\">edited</a> a pull request comment",
        "markdown": "Jamal Hartnett has [edited]
(https://fabrikam.visualstudio.com/DefaultCollection/_git/Fabrikam/pullreque
st/1?discussionId=5) a pull request comment"
    },
    "detailedMessage": {
        "text": "Jamal Hartnett has edited a pull request comment\r\nThis is
my comment.\r\n",
        "html": "Jamal Hartnett has <a
href=\"https://fabrikam.visualstudio.com/DefaultCollection/_git/Fabrikam/pul
lrequest/1?discussionId=5\">edited</a> a pull request comment<p>This is my
comment.</p>",
        "markdown": "Jamal Hartnett has [edited]
(https://fabrikam.visualstudio.com/DefaultCollection/_git/Fabrikam/pullreque
st/1?discussionId=5) a pull request comment\r\nThis is my comment.\r\n"
    },
    "resource": {
        "comment": {
            "id": 2,
            "parentCommentId": 1,
            "author": {
                "displayName": "Jamal Hartnett",
                "url":
"https://fabrikam.vssps.visualstudio.com/_apis/Identities/00000000-0000-
0000-0000-000000000000",
                "id": "00000000-0000-0000-0000-000000000000000000000-0000-
0000-0000-000000000000",
                "uniqueName": "fabrikamfiber4@hotmail.com",
                "imageUrl":
"https://fabrikam.visualstudio.com/DefaultCollection/_api/_common/identityIm
age?id=00000000-0000-0000-0000-000000000000"
            },
            "content": "This is my comment.",
            "publishedDate": "2024-06-17T11:22:33.456789Z",
            "lastUpdatedDate": "2024-06-17T16:58:33.123889Z",
            "lastContentUpdatedDate": "2024-06-17T16:58:33.123889Z",
            "commentType": "text",
            "_links": {
                "self": {
                    "href":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/pullRequests/1/threads/5/comments/2"
                },
                "repository": {
                    "href": "http://joscol2/DefaultCollection/ebed510c-62eb-
474b-965f-fd151ebb82e4/_apis/git/repositories/00000000-0000-0000-0000-
000000000000"
                },
                "threads": {
                    "href":
```

```
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/pullRequests/1/threads/5"
                }
            }
        },
        "pullRequest": {
            "repository": {
                "id": "00000000-0000-0000-0000-000000000000000000000-0000-
0000-0000-000000000000",
                "name": "Fabrikam",
                "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000",
                "project": {
                    "id": "00000000-0000-0000-0000-000000000000abcd-1234-
efgh-5678",
                    "name": "Fabrikam",
                    "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/projects/abcd-
1234-efgh-5678",
                    "state": "wellFormed",
                    "visibility": "unchanged",
                    "lastUpdateTime": "0001-01-01T00:00:00"
                },
                "defaultBranch": "refs/heads/master",
                "remoteUrl":
"https://fabrikam.visualstudio.com/DefaultCollection/_git/Fabrikam"
            },
            "pullRequestId": 1,
            "status": "active",
            "createdBy": {
                "displayName": "Jamal Hartnett",
                "url":
"https://fabrikam.vssps.visualstudio.com/_apis/Identities/00000000-0000-
0000-0000-000000000000",
                "id": "00000000-0000-0000-0000-000000000000000000000-0000-
0000-0000-000000000000",
                "uniqueName": "fabrikamfiber4@hotmail.com",
                "imageUrl":
"https://fabrikam.visualstudio.com/DefaultCollection/_api/_common/identityIm
age?id=00000000-0000-0000-0000-000000000000"
            },
            "creationDate": "2024-06-17T11:22:33.456789Z",
            "title": "my first pull request",
            "description": " - test2\r\n",
            "sourceRefName": "refs/heads/mytopic",
            "targetRefName": "refs/heads/master",
            "mergeStatus": "succeeded",
            "mergeId": "a1234567-abcde-1234-abc",
            "lastMergeSourceCommit": {
                "commitId": "00000000-0000-0000-0000-000000000000",
                "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
```

```
            },
            "lastMergeTargetCommit": {
                "commitId": "00000000-0000-0000-0000-000000000000",
                "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
            },
            "lastMergeCommit": {
                "commitId": "00000000-0000-0000-0000-000000000000",
                "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
            },
            "reviewers": [
                {
                    "reviewerUrl": null,
                    "vote": 0,
                    "displayName": "[Mobile]\\Mobile Team",
                    "url":
"https://fabrikam.vssps.visualstudio.com/_apis/Identities/00000000-0000-
0000-0000-000000000000",
                    "id": "00000000-0000-0000-0000-000000000000000000000-
0000-0000-0000-000000000000",
                    "uniqueName": "azure-
devops:///Classification/TeamProject/00000000-0000-0000-0000-
000000000000\\Mobile Team",
                    "imageUrl":
"https://fabrikam.visualstudio.com/DefaultCollection/_api/_common/identityIm
age?id=00000000-0000-0000-0000-000000000000",
                    "isContainer": true
                }
            ],
            "commits": [
                {
                    "commitId": "00000000-0000-0000-0000-000000000000",
                    "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/commits/00000000-0000-0000-0000-
000000000000"
                }
            ],
            "url":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
00000000-0000-0000-0000-000000000000/pullRequests/1",
            "_links": {
                "web": {
                    "href":
"https://fabrikam.visualstudio.com/DefaultCollection/_git/Fabrikam/pullreque
st/1#view=discussion"
                },
                "statuses": {
                    "href":
"https://fabrikam.visualstudio.com/DefaultCollection/_apis/git/repositories/
```

```
00000000-0000-0000-0000-000000000000/pullRequests/1/statuses"
                }
            }
        }
    },
    "resourceVersion": "2.0",
    "resourceContainers": {
        "collection": {
            "id": "00000000-0000-0000-0000-000000000000"
        },
        "account": {
            "id": "00000000-0000-0000-0000-000000000000"
        },
        "project": {
            "id": "00000000-0000-0000-0000-000000000000"
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```


Repository created

Event: A new repository is created.

- Publisher ID: `azure-devops`
- Event ID: `git.repository.created`
- Resource Name: `repository`

Settings

- `project`: Filter events to include only repositories created within the specified project.
- `repositoryName`: Filter events to include only repositories with the specified name pattern.

Sample payload

JSON


```
{
    "publisherId": "azure-devops",
    "eventId": "git.repository.created",
    "resource": {
        "repository": {
            "id": "00000000-0000-0000-0000-000000000000repository-id",
            "name": "repository-name",
            "url": "repository-url",
```

```
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            }
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```

Repository deleted

Event: A repository is deleted.

- Publisher ID: `azure-devops`
- Event ID: `git.repository.deleted`
- Resource Name: `repository`

Settings

- `project`: Filter events to include only repositories deleted within the specified project.
- `repositoryName`: Filter events to include only repositories with the specified name pattern.

Sample payload

JSON

```
{
    "publisherId": "azure-devops",
    "eventId": "git.repository.deleted",
    "resource": {
        "repository": {
            "id": "00000000-0000-0000-0000-000000000000repository-id",
            "name": "repository-name",
            "url": "repository-url",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            }
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```

Repository forked

Event: A repository is forked.

- Publisher ID: `azure-devops`
- Event ID: `git.repository.forked`
- Resource Name: `repository`

Settings

- `project`: Filter events to include only repositories forked within the specified project.
- `sourceRepositoryName`: Filter events to include only forks of repositories with the specified source repository name pattern.
- `forkedRepositoryName`: Filter events to include only repositories with the specified forked repository name pattern.

Sample payload

JSON


```
{
    "publisherId": "azure-devops",
    "eventId": "git.repository.forked",
    "resource": {
        "repository": {
            "id": "00000000-0000-0000-0000-000000000000forked-repository-
id",
            "name": "forked-repository-name",
            "url": "forked-repository-url",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            },
            "sourceRepository": {
                "id": "00000000-0000-0000-0000-000000000000source-
repository-id",
                "name": "source-repository-name",
                "url": "source-repository-url"
            }
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```

Repository renamed

Filter events to include only renamed repositories.

- Publisher ID: `azure-devops`
- Event ID: `git.repository.renamed`
- Resource Name: `repository`

Settings

- `project`: Filter events to include only repositories renamed within the specified project.
- `oldRepositoryName`: Filter events to include only repositories with the specified old name pattern.
- `newRepositoryName`: Filter events to include only repositories with the specified new name pattern.

Sample payload

JSON


```
{
    "publisherId": "azure-devops",
    "eventId": "git.repository.renamed",
    "resource": {
        "repository": {
            "id": "00000000-0000-0000-0000-000000000000repository-id",
            "oldName": "old-repository-name",
            "newName": "new-repository-name",
            "url": "repository-url",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            }
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```


Repository status changed

Event: A repository status is changed.

- Publisher ID: `azure-devops`

- Event ID: `git.repository.statusChanged`
- Resource Name: `repository`

Settings

- `project`: Filter events to include only repositories within the specified project.
- `repositoryName`: Filter events to include only repositories with the specified name pattern.
- `oldStatus`: Filter events to include only repositories with the specified old status.
- `newStatus`: Filter events to include only repositories with the specified new status.

Sample payload

JSON


```
{
    "publisherId": "azure-devops",
    "eventId": "git.repository.statusChanged",
    "resource": {
        "repository": {
            "id": "00000000-0000-0000-0000-000000000000repository-id",
            "name": "repository-name",
            "url": "repository-url",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            },
            "oldStatus": "old-status",
            "newStatus": "new-status"
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```


Service connection created

Event: A new service connection is created.

- Publisher ID: `azure-devops`
- Event ID: `serviceendpoint.created`
- Resource Name: `serviceendpoint`

Settings

- `project`: Filter events to include only service connections created within the specified project.
- `serviceConnectionType`: Filter events to include only service connections of the specified type.

Sample payload

JSON

```
{
    "publisherId": "azure-devops",
    "eventId": "serviceendpoint.created",
    "resource": {
        "serviceEndpoint": {
            "id": "00000000-0000-0000-0000-000000000000service-connection-
id",
            "name": "service-connection-name",
            "type": "service-connection-type",
            "url": "service-connection-url",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            }
        }
    },
    "createdDate": "2024-07-17T21:34:22.338Z"
}
```

Service connection updated

Event: A service connection is updated.

- Publisher ID: `azure-devops`
- Event ID: `serviceendpoint.updated`
- Resource Name: `serviceendpoint`

Settings

- `project`: Filter events to include only service connections updated within the specified project.
- `serviceConnectionType`: Filter events to include only service connections of the specified type.

Sample payload

JSON


```
{
    "publisherId": "azure-devops",
    "eventId": "serviceendpoint.updated",
    "resource": {
        "serviceEndpoint": {
            "id": "00000000-0000-0000-0000-000000000000service-connection-
id",
            "name": "service-connection-name",
            "type": "service-connection-type",
            "url": "service-connection-url",
            "project": {
                "id": "00000000-0000-0000-0000-000000000000project-id",
                "name": "project-name"
            }
        }
    },
    "updatedDate": "2024-07-17T21:34:22.338Z"
}
```


Work item

Work item created

Event: A new work item is created.

- Publisher ID: `azure-devops`
- Event ID: `workitem.created`
- Resource Name: `workitem`

Settings

- `areaPath`: Filter events to include only work items under the specified area path.
- `workItemType`: Filter events to include only work items of the specified type.

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000",
  "eventType": "workitem.created",
  "publisherId": "azure-devops",
  "scope": "all",
```

```
  "message": {
    "text": "Bug #5 (Some great new idea!) created by Jamal
Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=00000000-0000-0000-0000-000000000000)",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some
great new idea!) created by Jamal Hartnett.",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-
inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new
idea!) created by Jamal Hartnett."
  },
  "detailedMessage": {
    "text": "Bug #5 (Some great new idea!) created by Jamal
Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=00000000-0000-0000-0000-000000000000)\r\n\r\n- Area:
FabrikamCloud\r\n- Iteration: FabrikamCloud\\Release 1\\Sprint 1\r\n- State:
New\r\n- Assigned to: \r\n- Comment: \r\n- Severity: 3 - Medium\r\n",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some
great new idea!) created by Jamal Hartnett.<ul>\r\n- Area:
FabrikamCloud</br>\r\n- Iteration: FabrikamCloud\\Release 1\\Sprint
1</br>\r\n- State: New</br>\r\n- Assigned to: </br>\r\n- Comment: </br>\r\n-
Severity: 3 - Medium</br></ul>",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-
inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new
idea!) created by Jamal Hartnett.\r\n\r\n* Area: FabrikamCloud\r\n*
Iteration: FabrikamCloud\\Release 1\\Sprint 1\r\n* State: New\r\n* Assigned
to: \r\n* Comment: \r\n* Severity: 3 - Medium\r\n"
  },
  "resource": {
    "id": 5,
    "rev": 1,
    "fields": {
      "System.AreaPath": "FabrikamCloud",
      "System.TeamProject": "FabrikamCloud",
      "System.IterationPath": "FabrikamCloud\\Release 1\\Sprint 1",
      "System.WorkItemType": "Bug",
      "System.State": "New",
      "System.Reason": "New defect reported",
      "System.CreatedDate": "2014-07-15T17:42:44.663Z",
      "System.CreatedBy": "Jamal Hartnett",
      "System.ChangedDate": "2014-07-15T17:42:44.663Z",
      "System.ChangedBy": "Jamal Hartnett",
      "System.Title": "Some great new idea!",
      "Microsoft.Azure DevOps Services.Common.Severity": "3 - Medium",
      "WEF_EB329F44FE5F4A94ACB1DA153FDF38BA_Kanban.Column": "New"
    },
    "_links": {
      "self": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
      },
      "workItemUpdates": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/updates"
```

```
      },
      "workItemRevisions": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/revisions"
      },
      "workItemType": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/ea830882-2a3c-4095-a53f-
972f9a376f6e/workItemTypes/Bug"
      },
      "fields": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/fields"
      }
    },
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:29.7688022Z"
}
```


Work item deleted

Event: A work item is deleted.

- Publisher ID: `azure-devops`
- Event ID: `workitem.deleted`
- Resource Name: `resource`

Settings

- `areaPath`: Filter events to include only work items under the specified area path.
- `workItemType`: Filter events to include only work items of the specified type.

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-00000000000072da0ade-0709-40ee-beb7-
104287bf7e84",
  "eventType": "workitem.deleted",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Bug #5 (Some great new idea!) deleted by Jamal Hartnett.",
    "html": "Bug #5 (Some great new idea!) deleted by Jamal Hartnett.",
    "markdown": "[Bug #5](Some great new idea!) deleted by Jamal Hartnett."
  },
  "detailedMessage": {
    "text": "Bug #5 (Some great new idea!) deleted by Jamal
Hartnett.\r\n\r\n- Area: FabrikamCloud\r\n- Iteration:
FabrikamCloud\\Release 1\\Sprint 1\r\n- State: New\r\n",
    "html": "Bug #5 (Some great new idea!) deleted by Jamal Hartnett.
<ul>\r\n- Area: FabrikamCloud</br>\r\n- Iteration: FabrikamCloud\\Release
1\\Sprint 1</br>\r\n- State: New</br></ul>",
    "markdown": "[Bug #5](Some great new idea!) deleted by Jamal
Hartnett.\r\n\r\n* Area: FabrikamCloud\r\n* Iteration:
FabrikamCloud\\Release 1\\Sprint 1\r\n* State: New\r\n"
  },
  "resource": {
    "id": 5,
    "rev": 1,
    "fields": {
      "System.AreaPath": "FabrikamCloud",
      "System.TeamProject": "FabrikamCloud",
      "System.IterationPath": "FabrikamCloud\\Release 1\\Sprint 1",
      "System.WorkItemType": "Bug",
      "System.State": "New",
      "System.Reason": "New defect reported",
      "System.CreatedDate": "2014-07-15T17:42:44.663Z",
      "System.CreatedBy": "Jamal Hartnett",
      "System.ChangedDate": "2014-07-15T17:42:44.663Z",
      "System.ChangedBy": "Jamal Hartnett",
      "System.Title": "Some great new idea!",
      "Microsoft.Azure DevOps Services.Common.Severity": "3 - Medium",
      "WEF_EB329F44FE5F4A94ACB1DA153FDF38BA_Kanban.Column": "New"
    },
    "_links": {
      "self": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/recyclebin/5"
      },
      "workItemType": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/ea830882-2a3c-4095-a53f-
972f9a376f6e/workItemTypes/Bug"
      },
      "fields": {
        "href": "https://dev.azure.com/fabrikam-fiber-
```

```
inc/DefaultCollection/_apis/wit/fields"
      }
    },
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/recyclebin/5"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:30.0657064Z"
}
```


Work item restored

Event: A work item is newly restored.

- Publisher ID: `azure-devops`
- Event ID: `workitem.restored`
- Resource Name: `resource`

Settings

- `areaPath`: Filter events to include only work items under the specified area path.
- `workItemType`: Filter events to include only work items of the specified type.

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-0000000000001ca023d6-6cff-49dd-b3d1-
302b69311810",
  "eventType": "workitem.restored",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Bug #5 (Some great new idea!) restored by Jamal
Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
```

```
pcguid=00000000-0000-0000-0000-000000000000)",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some
great new idea!) restored by Jamal Hartnett.",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-
inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new
idea!) restored by Jamal Hartnett."
  },
  "detailedMessage": {
    "text": "Bug #5 (Some great new idea!) restored by Jamal
Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=00000000-0000-0000-0000-000000000000)\r\n\r\n- Area:
FabrikamCloud\r\n- Iteration: FabrikamCloud\\Release 1\\Sprint 1\r\n- State:
New\r\n- Severity: 3 - Medium\r\n",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some
great new idea!) restored by Jamal Hartnett.<ul>\r\n- Area:
FabrikamCloud</br>\r\n- Iteration: FabrikamCloud\\Release 1\\Sprint
1</br>\r\n- State: New</br>Severity: 3 - Medium</br></ul>",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-
inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new
idea!) restored by Jamal Hartnett.\r\n\r\n* Area: FabrikamCloud\r\n*
Iteration: FabrikamCloud\\Release 1\\Sprint 1\r\n* State: New\r\n* Severity:
3 - Medium\r\n"
  },
  "resource": {
    "id": 5,
    "rev": 1,
    "fields": {
      "System.AreaPath": "FabrikamCloud",
      "System.TeamProject": "FabrikamCloud",
      "System.IterationPath": "FabrikamCloud\\Release 1\\Sprint 1",
      "System.WorkItemType": "Bug",
      "System.State": "New",
      "System.Reason": "New defect reported",
      "System.CreatedDate": "2014-07-15T17:42:44.663Z",
      "System.CreatedBy": "Jamal Hartnett",
      "System.ChangedDate": "2014-07-15T17:42:44.663Z",
      "System.ChangedBy": "Jamal Hartnett",
      "System.Title": "Some great new idea!",
      "Microsoft.Azure DevOps Services.Common.Severity": "3 - Medium",
      "WEF_EB329F44FE5F4A94ACB1DA153FDF38BA_Kanban.Column": "New"
    },
    "_links": {
      "self": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
      },
      "workItemUpdates": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/updates"
      },
      "workItemRevisions": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/revisions"
```

```
      },
      "workItemType": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/ea830882-2a3c-4095-a53f-
972f9a376f6e/workItemTypes/Bug"
      },
      "fields": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/fields"
      },
      "html": {
        "href": "https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=d81542e4-cdfa-4333-b082-1ae2d6c3ad16&id=5"
      },
      "workItemHistory": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/history"
      }
    },
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:30.1456784Z"
}
```


Work item updated

Event: A work item is changed.

- Publisher ID: `azure-devops`
- Event ID: `workitem.updated`
- Resource Name: `workitem`

> **Note**
>
> You can't create a Service Hooks subscription with multiple fields through the UI. But, you can do so in the following ways:

- Create a custom payload through the API
- Create separate Service Hooks subscriptions for each field

Settings

- `areaPath`: Filter events to include only work items under the specified area path.
- `changedFields`: Filter events to include only work items with the specified field changed.
- `workItemType`: Filter events to include only work items of the specified type.

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-0000000000001ca023d6-6cff-49dd-b3d1-
302b69311810",
  "eventType": "workitem.updated",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Bug #5 (Some great new idea!) updated by Jamal
Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=00000000-0000-0000-0000-000000000000)",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some
great new idea!) updated by Jamal Hartnett.",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-
inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new
idea!) updated by Jamal Hartnett."
  },
  "detailedMessage": {
    "text": "Bug #5 (Some great new idea!) updated by Jamal
Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=00000000-0000-0000-0000-000000000000)\r\n\r\n- Area:
FabrikamCloud\r\n- Iteration: FabrikamCloud\\Release 1\\Sprint 1\r\n- State:
New\r\n- Severity: 3 - Medium\r\n",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some
great new idea!) updated by Jamal Hartnett.<ul>\r\n- Area:
FabrikamCloud</br>\r\n- Iteration: FabrikamCloud\\Release 1\\Sprint
1</br>\r\n- State: New</br>Severity: 3 - Medium</br></ul>",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-
inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new
idea!) updated by Jamal Hartnett.\r\n\r\n* Area: FabrikamCloud\r\n*
Iteration: FabrikamCloud\\Release 1\\Sprint 1\r\n* State: New\r\n* Severity:
3 - Medium\r\n"
  },
```

```
  "resource": {
    "id": 5,
    "rev": 1,
    "fields": {
      "System.AreaPath": "FabrikamCloud",
      "System.TeamProject": "FabrikamCloud",
      "System.IterationPath": "FabrikamCloud\\Release 1\\Sprint 1",
      "System.WorkItemType": "Bug",
      "System.State": "New",
      "System.Reason": "New defect reported",
      "System.CreatedDate": "2014-07-15T17:42:44.663Z",
      "System.CreatedBy": "Jamal Hartnett",
      "System.ChangedDate": "2014-07-15T17:42:44.663Z",
      "System.ChangedBy": "Jamal Hartnett",
      "System.Title": "Some great new idea!",
      "Microsoft.Azure DevOps Services.Common.Severity": "3 - Medium",
      "WEF_EB329F44FE5F4A94ACB1DA153FDF38BA_Kanban.Column": "New"
    },
    "_links": {
      "self": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
      },
      "workItemUpdates": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/updates"
      },
      "workItemRevisions": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/revisions"
      },
      "workItemType": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/ea830882-2a3c-4095-a53f-
972f9a376f6e/workItemTypes/Bug"
      },
      "fields": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/fields"
      },
      "html": {
        "href": "https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?
pcguid=d81542e4-cdfa-4333-b082-1ae2d6c3ad16&id=5"
      },
      "workItemHistory": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/history"
      }
    },
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
```

```
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:30.1456784Z"
}
```


Create a service hooks subscription

Do the following steps to create a service hooks subscription for each work item field.

1. Go to your **Project settings**.
2. Select **Service Hooks** > + **Create subscription**.
3. Choose **Web Hooks** as the type of event > **Next**.
4. Select **Work item updated** as the type of trigger event.
5. Specify the field you want to monitor for changes in the subscription configuration dialog. For example, if you want to track changes in the **State** field, set the filter to **Field: State**. You can also filter events to include only items under a specified area path, only specified work item types, and only work items containing a specified tag.
6. Select **Next**.
7. Continue in the SETTINGS dialog, and then **Test** or **Finish** creating your subscription.

Work item commented on

Event: A work item is commented on.

- Publisher ID: `azure-devops`
- Event ID: `workitem.commented`
- Resource Name: `workitem`

Settings

- `areaPath`: Filter events to include only work items under the specified area path.
- `commentPattern`: The string that must be found in the comment.
- `workItemType`: Filter events to include only work items of the specified type.

Sample payload

JSON


```
{
  "id": "00000000-0000-0000-0000-000000000000fb2617ed-60df-4518-81fa-749faa6c5cd6",
  "eventType": "workitem.commented",
  "publisherId": "azure-devops",
  "scope": "all",
  "message": {
    "text": "Bug #5 (Some great new idea!) commented on by Jamal Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000)",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some great new idea!) commented on by Jamal Hartnett.",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new idea!) commented on by Jamal Hartnett."
  },
  "detailedMessage": {
    "text": "Bug #5 (Some great new idea!) commented on by Jamal Hartnett.\r\n(https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000)\r\nThis is a great new idea",
    "html": "<a href=\"https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?pcguid=74e918bf-3376-436d-bd20-8e8c1287f465&amp;id=5\">Bug #5</a> (Some great new idea!) commented on by Jamal Hartnett.<br/>This is a great new idea",
    "markdown": "[Bug #5](https://dev.azure.com/fabrikam-fiber-inc/web/wi.aspx?pcguid=00000000-0000-0000-0000-000000000000) (Some great new idea!) commented on by Jamal Hartnett.\r\nThis is a great new idea"
  },
  "resource": {
    "id": 5,
    "rev": 4,
    "fields": {
      "System.AreaPath": "FabrikamCloud",
      "System.TeamProject": "FabrikamCloud",
      "System.IterationPath": "FabrikamCloud\\Release 1\\Sprint 1",
      "System.WorkItemType": "Bug",
      "System.State": "New",
      "System.Reason": "New defect reported",
      "System.CreatedDate": "2014-07-15T17:42:44.663Z",
      "System.CreatedBy": "Jamal Hartnett",
      "System.ChangedDate": "2014-07-15T17:42:44.663Z",
      "System.ChangedBy": "Jamal Hartnett",
      "System.Title": "Some great new idea!",
      "Microsoft.Azure DevOps Services.Common.Severity": "3 - Medium",
      "WEF_EB329F44FE5F4A94ACB1DA153FDF38BA_Kanban.Column": "New",
      "System.History": "This is a great new idea"
    },
    "_links": {
      "self": {
```

```
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
      },
      "workItemUpdates": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/updates"
      },
      "workItemRevisions": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5/revisions"
      },
      "workItemType": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/00-00-00/workItemTypes/Bug"
      },
      "fields": {
        "href": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/fields"
      }
    },
    "url": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection/_apis/wit/workItems/5"
  },
  "resourceVersion": "1.0",
  "resourceContainers": {
    "collection": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "account": {
      "id": "00000000-0000-0000-0000-000000000000"
    },
    "project": {
      "id": "00000000-0000-0000-0000-000000000000"
    }
  },
  "createdDate": "2024-09-19T13:03:28.9695265Z"
}
```


Resource containers

The event payload contains a `resourceContainers` dictionary that includes the IDs of the project, collection/account, or server that the event initiated from.

Some products/environments also include a `baseUrl` field with each entry that provides the full URL to the container. You can use this URL to create a connection to the container to make REST API calls.

ⓘ **Note:** The author created this article with assistance from AI. Learn more

Feedback

Was this page helpful? Yes No

Provide product feedback

Troubleshoot service hooks

Article • 03/25/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

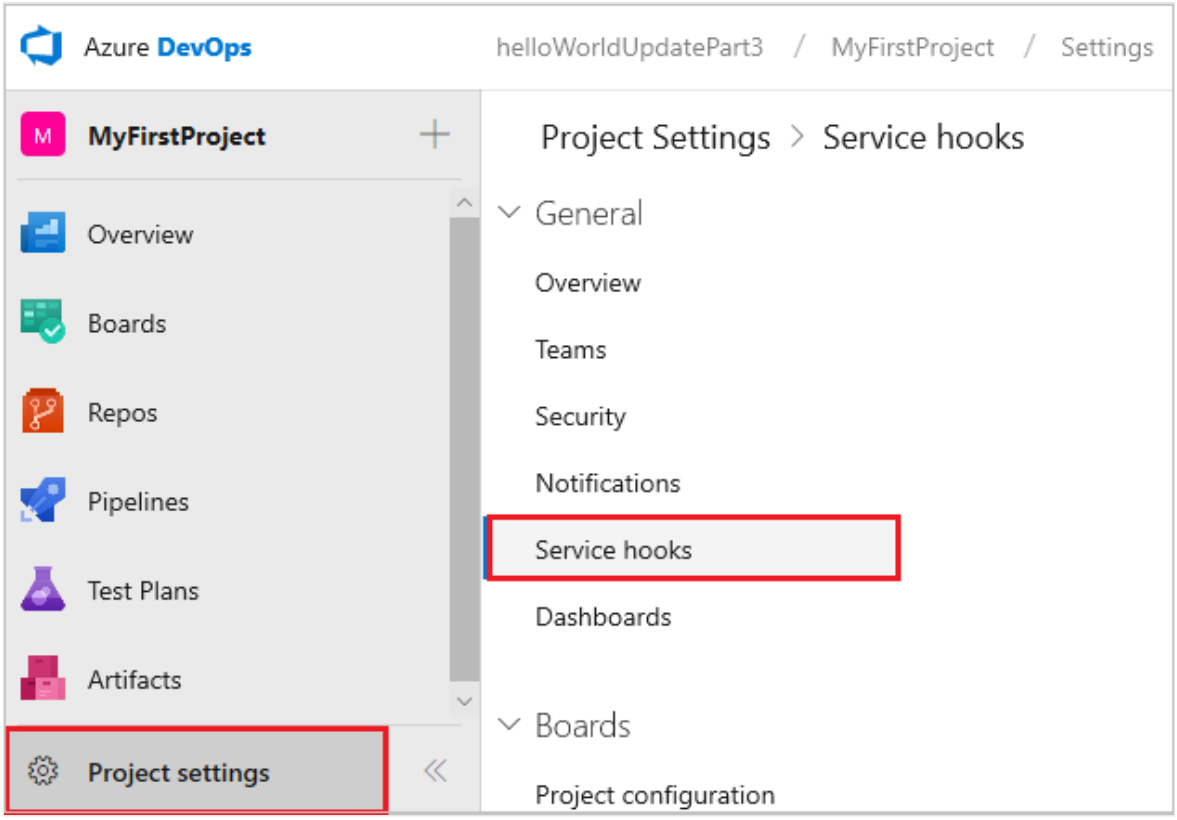
Use this article for general troubleshooting guidance and answers to frequently asked questions (FAQs).

View activity and debug problems

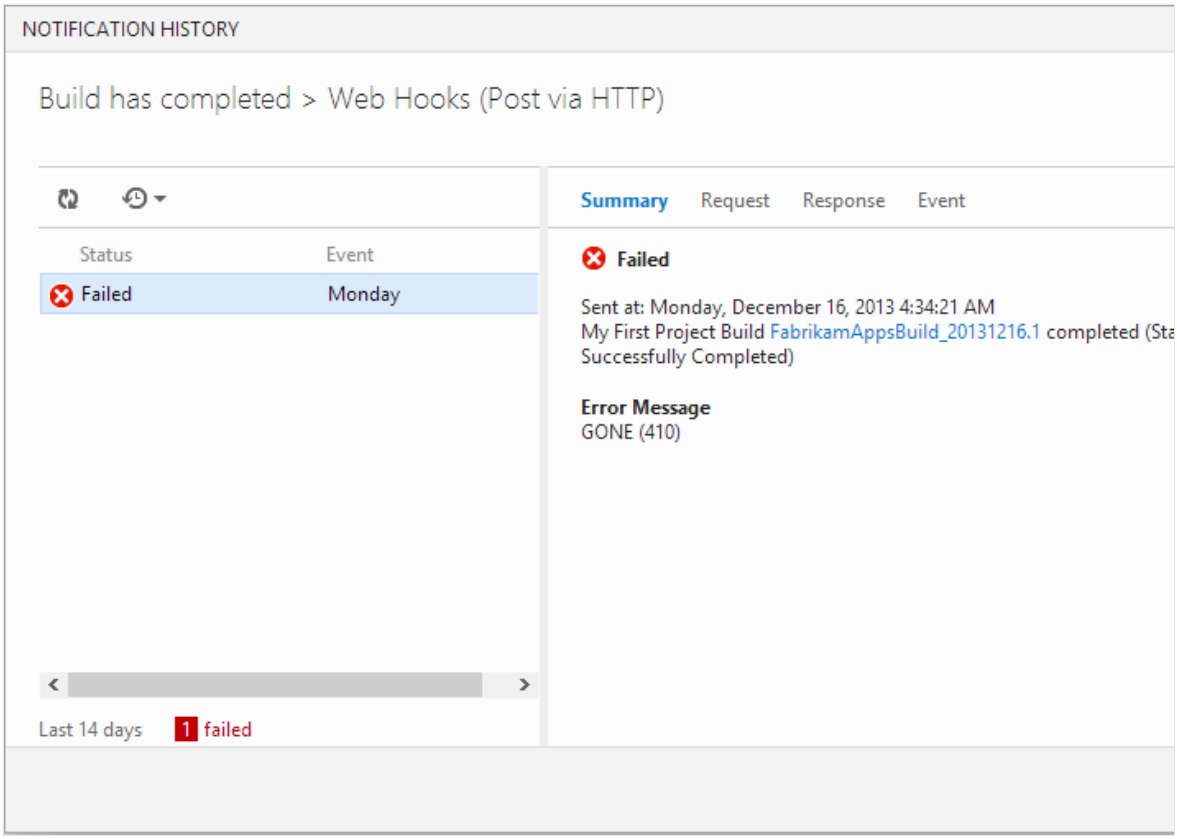
The **Service Hooks** page in the web access admin shows your recent activity (last 14 days) for each subscription, and whether a subscription is enabled, disabled, or restricted.

You can access detailed history about a subscription including detailed request/response data, which is useful for debugging a problematic service or subscription.

1. To view the activity and status of your subscriptions, go to the **Service Hooks** page.



2. To view detailed activity for a subscription, including full request, response, and event payload data, select a subscription in the table, and then select **History**.



Subscription failures and probation (restricted)

Failure types

Failures from a Service Hooks notification are grouped into the following categories:

- Terminal Failures
- Transient Failures
- Enduring Failures

Terminal Failures

The only Terminal Failure is HTTP Status Code 410 (Gone). When a subscription sees a Terminal Failure, it's automatically disabled no matter its prior status.

Transient Failures

When a subscription sees a Transient Failure, it attempts to resend the notification up to eight times, with an increasing delay between each attempt. Transient Failures include the following codes:

- 408 (Request Timeout)
- 502 (Bad Gateway)

- 503 (Service Unavailable)
- 504 (Gateway Timeout)

Sequence of retries for transient failures

Expand table

Retry #	Wait time
Before retry 1	wait ~1 second
Before retry 2	wait ~2 seconds (total delay of 3 seconds)
Before retry 3	wait ~4 seconds (total delay of 7 seconds)
Before retry 4	wait ~8 seconds (total delay of 15 seconds)
Before retry 5	wait ~16 seconds (total delay of 31 seconds)
Before retry 6	wait ~32 seconds (total delay of 63 seconds)
Before retry 7	wait ~60 seconds (max backoff time, total delay of 123 seconds)
Before retry 8	wait ~60 seconds (max backoff time, total delay of 183 seconds)

If the notification exhausts all of its retries and continues to see a Transient Failure for each attempt, the subscription stops trying to send the notification, and treats the notification as if it saw an Enduring Failure.

Enduring Failures

Enduring Failures include all other HTTP failure codes, for example: 404 (Not Found), 500 (Internal Server Error), and so on.

When a subscription sees an Enduring Failure, it's placed on probation.

Probation

While on probation, a subscription is limited in the number of notifications it can send. If the subscription continues to hit Enduring Failures, then it gets increasingly limited, and eventually disabled. If the subscription receives a successful response while on probation, it gets restored to a fully enabled state.

Sequence of seven maximum retries while subscription is on probation

When a subscription is in probation, any new events are lost. Once a retry is successful, the subscription is enabled, and events are published again.

Expand table

Retry #	Wait time
Before retry 1	wait ~20 minutes
Before retry 2	wait ~40 minutes (total probation time of 1 hour)
Before retry 3	wait ~1 hour 20 minutes (total probation time of 2.33 hours)
Before retry 4	wait ~2 hours 40 minutes (total probation time of 5 hours)
Before retry 5	wait ~5 hours 20 minutes (total probation time of 10.33 hours)
Before retry 6	wait ~10 hours 40 minutes (total probation time of 21 hours)
Before retry 7	wait ~15 hours (max backoff time, total probation time of 36 hours)

After seven retries, the subscription status gets set to *DisabledBySystem* if notifying the consumer fails.

FAQs

Q: What is the payload limit of a service-hook?

A: The payload limit is 2 MB. Larger payloads cause degradation in performance and reliability. As a best practice, service-hooks should limit the payload to 2 MB or less.

Q: What does the status Enabled (restricted) mean?

A: A subscription becomes restricted if too many failures occur. Enabled (restricted) is the same as being on probation.

Q: What does the status Disabled (due to failures) mean?

A: A subscription is automatically disabled after a series of consecutive failures over a prolonged period or a *terminal failure* is encountered. *Transient failures* types are retried

several times before being declared a failure. *Enduring failure* types aren't retried. The following are examples of each type of failure.

- Transient: 408 (Request Timeout), 502 (Bad Gateway), 503 (Service Unavailable), 504 (Gateway Timeout)
- Terminal: 410 (Gone)
- Enduring: All failures that aren't transient or terminal

Q: What does the status Disabled (user left project) mean?

A: The user who created the subscription is no longer a member of the team.

Q: What should I try if a service hook isn't working?

A: Check the following items:

- Confirm the subscription is enabled
- Confirm the subscription settings are correct (both event filters and actions)
- Look at the history, especially if there are failures

Q: Can I grant a regular project user the ability to view and manage service hook subscriptions for a project?

A: By default, only project administrators have these permissions. To grant them to other users directly, you can use the command line tool or the Security REST API.

Q: Can I programmatically create subscriptions?

A: Yes, use REST APIs.

Feedback

Was this page helpful? Yes No

Provide product feedback

Integrate applications

Article • 01/29/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

You can build custom applications or services that integrate with Azure DevOps by using the **REST APIs** to make direct HTTP calls, or use our **.NET Client Libraries**.

Along with interacting with Azure DevOps in your application, you can also integrate with popular non-Microsoft services such as Slack or Jenkins.

To get started, see the following articles:

- Create a bug
- Get work items using queries

Integrate applications with Azure DevOps

Integrate Azure DevOps with other tools and services to not only enhance team collaboration but also streamline workflows across the software development lifecycle. Whether you're building custom integrations or using off-the-shelf services, it's essential that you understand the prerequisites.

Prerequisites for integration

Before you start the integration process, ensure you have the following items:

- **Access:** Ensure proper permissions and use secure authentication.
- **API knowledge:** Understand Azure DevOps REST API and the target tool API.
- **Data mapping:** Identify and align fields and data types between systems to ensure compatibility in field names, formats, and value ranges.
- **Network and connectivity:** Ensure your network supports API calls, webhook notifications, and connectivity between Azure DevOps and self-hosted tools.

Resources

- REST API reference

Feedback

Was this page helpful? Yes No

Provide product feedback

Create a bug in Azure DevOps Services using .NET client libraries

Article • 05/30/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Creating a new bug (or any work item) is pretty straight forward. You just need to set the field values and send a JSON-Patch object to the REST endpoint.

Prerequisites

You must have the following before you can create a work item.

- An organization in Azure DevOps Services. If you don't have one, you can create one for free
- A Personal Access Token, find out how to create one
- A C# development environment, you can use Visual Studio

Create a C# project in Visual Studio

To learn about C# programming within Visual Studio, find the Visual Studio C# programming documentation

C# code content

There are a few things happening in the following code sample:

1. Authentication
   a. Creating credentials using your PAT
   b. Creating a VSSConnection with your Azure DevOps Services URI and the credentials
2. Retrieving the client using your VSSConnection
3. Creating the bug
   a. Create an array of objects to set the field values
   b. Convert that array to a serialized json object
   c. Send that serialized json object to the REST endpoint

C# code snippet

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using Microsoft.TeamFoundation.WorkItemTracking.WebApi;
using Microsoft.TeamFoundation.WorkItemTracking.WebApi.Models;
using Microsoft.VisualStudio.Services.Common;
using Microsoft.VisualStudio.Services.WebApi.Patch.Json;
using Microsoft.VisualStudio.Services.WebApi.Patch;
using Microsoft.VisualStudio.Services.WebApi;
using System.Net.Http.Headers;
using System.Net.Http;
using Newtonsoft.Json;


public class CreateBug
{
    readonly string _uri;
    readonly string _personalAccessToken;
    readonly string _project;

    /// <summary>
    /// Constructor. Manually set values to match your organization.
    /// </summary>
    public CreateBug()
    {
        _uri = "https://dev.azure.com/{orgName}";
        _personalAccessToken = "personal access token";
        _project = "project name";
    }

    /// <summary>
    /// Create a bug using the .NET client library
    /// </summary>
    ///
<returns>Microsoft.TeamFoundation.WorkItemTracking.WebApi.Models.WorkItem</r
eturns>
    public WorkItem CreateBugUsingClientLib()
    {
        Uri uri = new Uri(_uri);
        string personalAccessToken = _personalAccessToken;
        string project = _project;

        VssBasicCredential credentials = new VssBasicCredential("",
_personalAccessToken);
        JsonPatchDocument patchDocument = new JsonPatchDocument();
```

```
        //add fields and their values to your patch document
        patchDocument.Add(
            new JsonPatchOperation()
            {
                Operation = Operation.Add,
                Path = "/fields/System.Title",
                Value = "Authorization Errors"
            }
        );

        patchDocument.Add(
            new JsonPatchOperation()
            {
                Operation = Operation.Add,
                Path = "/fields/Microsoft.VSTS.TCM.ReproSteps",
                Value = "Our authorization logic needs to allow for users
with Microsoft accounts (formerly Live Ids) - http://
msdn.microsoft.com/library/live/hh826547.aspx"
            }
        );

        patchDocument.Add(
            new JsonPatchOperation()
            {
                Operation = Operation.Add,
                Path = "/fields/Microsoft.VSTS.Common.Priority",
                Value = "1"
            }
        );

        patchDocument.Add(
            new JsonPatchOperation()
            {
                Operation = Operation.Add,
                Path = "/fields/Microsoft.VSTS.Common.Severity",
                Value = "2 - High"
            }
        );
        VssConnection connection = new VssConnection(uri, credentials);
        WorkItemTrackingHttpClient workItemTrackingHttpClient =
connection.GetClient<WorkItemTrackingHttpClient>();

        try
        {
            WorkItem result =
workItemTrackingHttpClient.CreateWorkItemAsync(patchDocument, project,
"Bug").Result;

            Console.WriteLine("Bug Successfully Created: Bug #{0}",
result.Id);

            return result;
        }
        catch (AggregateException ex)
        {
```

```
            Console.WriteLine("Error creating bug: {0}",
ex.InnerException.Message);
            return null;
        }
    }
}
```

Next steps

- Check out another Quickstart: Get a list of work items using queries
- Explore the integrate samples

Fetch work items with queries programmatically

Article • 10/03/2024

Azure DevOps Services

Fetching work items using queries is a common scenario in Azure DevOps Services. This article explains how to implement this scenario programmatically using REST APIs or .NET client libraries.

Prerequisites

- **Azure DevOps setup:**
  - Have an organization in Azure DevOps Services.
  - Have a Personal Access Token (PAT).
- **Development environment:** Have a C# development environment. You can use Visual Studio.

Create a C# project in Visual Studio

For information about C# programming within Visual Studio, see the Visual Studio C# programming documentation.

C# code content

The following tasks occur in the code snippet:

- **Authenticate**
  1. Create credentials using your Personal Access Token (PAT).
  2. Generate the client using the credentials.
- **Get the work items**
  1. Create the query you want to use.
  2. Retrieve the results for that query.
  3. Fetch each of the work items by ID.

C# code snippet

cs

```cs
// nuget:Microsoft.TeamFoundationServer.Client
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.TeamFoundation.WorkItemTracking.WebApi;
using Microsoft.TeamFoundation.WorkItemTracking.WebApi.Models;
using Microsoft.VisualStudio.Services.Common;
using Microsoft.VisualStudio.Services.WebApi;

public class QueryExecutor
{
    private readonly Uri uri;
    private readonly string personalAccessToken;

    /// <summary>
    /// Initializes a new instance of the <see cref="QueryExecutor" />
class.
    /// </summary>
    /// <param name="orgName">
    /// An organization in Azure DevOps Services. If you don't have one, you
can create one for free:
    /// <see href="https://go.microsoft.com/fwlink/?LinkId=307137" />.
    /// </param>
    /// <param name="personalAccessToken">
    /// A Personal Access Token, find out how to create one:
    /// <see href="/azure/devops/organizations/accounts/use-personal-access-
tokens-to-authenticate?view=azure-devops" />.
    /// </param>
    public QueryExecutor(string orgName, string personalAccessToken)
    {
        this.uri = new Uri("https://dev.azure.com/" + orgName);
        this.personalAccessToken = personalAccessToken;
    }

    /// <summary>
    /// Execute a WIQL (Work Item Query Language) query to return a list of
open bugs.
    /// </summary>
    /// <param name="project">The name of your project within your
organization.</param>
    /// <returns>A list of <see cref="WorkItem"/> objects representing all
the open bugs.</returns>
    public async Task<IList<WorkItem>> QueryOpenBugs(string project)
    {
        var credentials = new VssBasicCredential(string.Empty,
this.personalAccessToken);
        var wiql = new Wiql()
        {
            Query = "Select [Id] " +
                    "From WorkItems " +
                    "Where [Work Item Type] = 'Bug' " +
```

```
                    "And [System.TeamProject] = '" + project + "' " +
                    "And [System.State] <> 'Closed' " +
                    "Order By [State] Asc, [Changed Date] Desc",
        };

        using (var httpClient = new WorkItemTrackingHttpClient(this.uri, new
VssCredentials(credentials)))
        {
            try
            {
                var result = await
httpClient.QueryByWiqlAsync(wiql).ConfigureAwait(false);
                var ids = result.WorkItems.Select(item =>
item.Id).ToArray();

                if (ids.Length == 0)
                {
                    return Array.Empty<WorkItem>();
                }

                var fields = new[] { "System.Id", "System.Title",
"System.State" };
                return await httpClient.GetWorkItemsAsync(ids, fields,
result.AsOf).ConfigureAwait(false);
            }
            catch (Exception ex)
            {
                Console.WriteLine("Error querying work items: " +
ex.Message);
                return Array.Empty<WorkItem>();
            }
        }
    }

    /// <summary>
    /// Execute a WIQL (Work Item Query Language) query to print a list of
open bugs.
    /// </summary>
    /// <param name="project">The name of your project within your
organization.</param>
    /// <returns>An async task.</returns>
    public async Task PrintOpenBugsAsync(string project)
    {
        var workItems = await
this.QueryOpenBugs(project).ConfigureAwait(false);
        Console.WriteLine("Query Results: {0} items found",
workItems.Count);

        foreach (var workItem in workItems)
        {
            Console.WriteLine(
                "{0}\t{1}\t{2}",
                workItem.Id,
                workItem.Fields["System.Title"],
                workItem.Fields["System.State"]);
```

```
        }
    }
}
```

Related articles

- Create a bug
- Integrate samples

ⓘ **Note:** The author created this article with assistance from AI. Learn more

---

Feedback

Was this page helpful? Yes No

Provide product feedback

Best integration practices

Article • 11/10/2023

Azure DevOps Services

Tools and integrations between services improve efficiencies for Azure DevOps Services. If you aren't careful, automated tools can get out of control executing a high rate of requests. The requests cause Azure DevOps to enforce rate limits to your organization. To help reduce your risk of hitting the rate limits, follow these best practices when you're using the REST APIs to integrate with Azure DevOps.

Push only actionable work items

Only push actionable items into Azure DevOps that your team plans to engage on or address in the future. Keep work items out of Azure DevOps until necessary. For example, don't attempt to store telemetry data in Azure DevOps.

Maintain your own data store

Don't add work items into Azure DevOps for the sake of having them all in one place. Azure DevOps Services isn't designed as a data storage service. Maintain your own data store.

Batch your changes

Doing single operations is slow and expensive, which is the leading cause for performance issues and rate limiting. Batch your changes into a single call. For more information, see our batch documentation and sample code.

Limit your revisions

Many revisions on a single work item create bloat and cause performance problems. We recommend doing the following tasks:

- Reduce your updates by batching your field changes. Don't update just one field at a time.
- If you have changes to multiple work items, batch those changes into a single action.
- Keep the number of revisions to a minimum to avoid revision limits.

> **Note**
>
> A work item revision limit of 10,000 is in effect for updates made through the REST API. This limit restricts updates from the REST API, but, updates from the web portal aren't affected.

Optimize queries

Optimize your queries to return a modest number of results. Complex conditions and filters can lead to long-running queries. Keep your queries execution time under 30 seconds to avoid threshold failures.

Query performance tips

- Place a date or range-limiting clause near the top of a query whenever possible.
- Reduce the number of clauses that use the *Ever* operator.
- Reduce the number of clauses that use the *Contains* operator, except for Tags.
  - Use the *Contains Words* operator when available.
  - Don't use the *Contains* operator on long text fields, as it's expensive.
- Avoid the '<>' and not operators when possible.
- Avoid using the *In Group* operator for large groups.
- Minimize the number of Or operators and ensure you still have top-level scoping before using.
- Avoid using an *OR* clause between an *In Group* operator and Area or Iteration Paths.
- Reduce the number of overall clauses to achieve your goal when possible.
- Avoid sorting on anything other than core fields, such as *ID*, when possible.
- Use a custom field in your filters if you want to sort on a custom field.
- Specify a project if possible. Otherwise, the query gets scoped to the entire collection and could take dramatically longer than it needs to. Uncheck "Query across projects on the top-right corner" of the query editor.

Query across projects

- Specify which project you're looking for if the query requires search across projects.
- Use *tags* instead of *keywords* when possible, unless you're searching for partial text of a string.

Handle failures gracefully

Updates and queries fail when resource limits or frequency of utilization crosses the limit threshold. For example, a query that runs longer than 30 seconds returns the following error:

```
VS402335: The timeout period (30 seconds) elapsed prior to completion of the query or the server is not responding.
```

When you're consuming the REST APIs, make sure you design your code to handle failures appropriately.

Limit your links

Limit the number of links per work item as much as possible, to avoid enforcement of link limits.

> **Important**
>
> We plan to enforce work item revision and link limits in the near future. These limits get determined by performance monitoring and customer feedback.

Don't use queries for reporting

Using queries and individual *get work item* calls is the top way to get rate limits enforced on your organization. Don't execute queries to return large lists of work items. Use the reporting work item links and work item revisions REST APIs instead.

For more information, see our C# sample on GitHub.

Related articles

- .NET client libraries
- Migration guide
- Service hooks
- REST API versioning

Service hooks in Azure DevOps Services REST APIs

Article • 11/10/2023

Azure DevOps Services

When you use the Subscription REST APIs, you can programmatically create a subscription that performs an action on an external (consumer) service when a specific event occurs in a project. For example, you can create a subscription to notify your service when a build fails.

Supported events:

- Build completed
- Code pushed (for Git projects)
- Pull request create or updated (for Git projects)
- Code checked in (TFVC projects)
- Work item created, updated, deleted, restored or commented on

You can configure filters on your subscriptions to control which events trigger an action. For example, you can filter the build completed event based on the build status. For a complete set of supported events and filter options, see the event reference.

For a complete set of supported consumer services and actions, see the consumer reference.

Create a service hook subscription programmatically

Article • 10/17/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Using the Subscriptions REST APIs , you can programmatically create a subscription that performs an action on an external/consumer service when a specific event occurs in an Azure DevOps project. For example, you can create a subscription to notify your service when a build fails.

Supported events:

- Build completed
- Code pushed (for Git projects)
- Pull request create or updated (for Git projects)
- Code checked in (TFVC projects)
- Work item created, updated, deleted, restored or commented on

You can configure filters on your subscriptions to control which events trigger an action. For example, you can filter the build completed event based on the build status. For a complete set of supported events and filter options, see the Event reference.

For a complete set of supported consumer services and actions, see the Consumer reference.

Prerequisites

To create a subscription, the following data is required:

- Project ID. Use the Project REST API to get the project ID.
- Event ID and settings. See the Event reference.
- Consumer and action IDs and settings. See the Consumer reference.

Create the request

Construct the body of the HTTP POST request to create the subscription based on the project ID, event, consumer, and action.

See the following example request for creating a subscription that causes a build event to POST to `https://myservice/event` when the build `WebSite.CI` fails.

Request

JavaScript

```
{
    "publisherId": "tfs",
    "eventType": "build.complete",
    "resourceVersion": "1.0",
    "consumerId": "webHooks",
    "consumerActionId": "httpRequest",
    "publisherInputs": {
        "buildStatus": "failed",
        "definitionName": "WebSite.CI",
        "projectId": "56479caf-1eeb-4bca-86ab-aaa6f29399d9",
    },
    "consumerInputs": {
        "url": " https://myservice/event"
    },
}
```

We highly recommend using secure HTTPS URLs for the security of the private data in the JSON object.

Response See the following response to the request to create the subscription:

JavaScript

```
{
    "id": "74aeeed0-bf5d-48dc-893f-f862b80987e9",
    "url":
"https://dev.azure.com/fabrikam/DefaultCollection/_apis/hooks/subscriptions/
74aeeed0-bf5d-48dc-893f-f862b80987e9",
    "publisherId": "tfs",
    "eventType": "build.complete",
    "resourceVersion": "1.0",
    "consumerId": "webHooks",
    "consumerActionId": "httpRequest",
    "createdBy": {
        "id": "00ca946b-2fe9-4f2a-ae2f-40d5c48001bc"
    },
    "createdDate": "2014-03-28T16:10:06.523Z",
    "modifiedBy": {
        "id": "1c4978ae-7cc9-4efa-8649-5547304a8438"
    },
    "modifiedDate": "2014-04-25T18:15:26.053Z",
    "publisherInputs": {
        "buildStatus": "failed",
```

```
        "definitionName": "WebSite.CI",
        "hostId": "17f27955-99bb-4861-9550-f2c669d64fc9",
        "projectId": "56479caf-1eeb-4bca-86ab-aaa6f29399d9",
        "tfsSubscriptionId": "29cde8b4-f37e-4ef9-a6d4-d57d526d82cc"
    },
    "consumerInputs": {
        "url": "http://myservice/event"
    }
}
```

If the subscription request fails, you get an HTTP response code of 400 with a message that has further details.

What happens when the event occurs?

When an event occurs, all enabled subscriptions in the project are evaluated, and the consumer action is performed for all matching subscriptions.

Resource versions (advanced)

Resource versioning is applicable when an API is in preview. For most scenarios, specifying `1.0` as the resource version is the safest route.

The event payload sent to certain consumers, like Webhooks, Azure Service Bus, and Azure Storage, includes a JSON representation of subject resource (for example, a build or work item). The representation of this resource can have different forms or versions.

You can specify the version of the resource that you want to have sent to the consumer service via the `resourceVersion` field on the subscription. The resource version is the same as the API version. Not specifying a resource version means "latest released". You should always specify a resource version, which ensures a consistent event payload over time.

FAQs

Q: Are there services that I can subscribe to manually?

A: Yes. For more information about the services that you can subscribe to from the administration page for a project, see the Overview.

Q: Are there C# libraries that I can use to create subscriptions?

A: No, but here's a sample to help you get started.

C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Mvc;

namespace Microsoft.Samples.VisualStudioOnline
{
    public class ServiceHookEventController : Controller
    {

        // POST: /ServiceHookEvent/workitemcreated
        [HttpPost]
        public HttpResponseMessage WorkItemCreated(Content workItemEvent)
        {
            //Grabbing the title for the new workitem
            var value = RetrieveFieldValue("System.field",
workItemEvent.Resource.Fields);

            //Acknowledge event receipt
            return new HttpResponseMessage(HttpStatusCode.OK);
        }

        /// <summary>
        /// Gets the value for a specified work item field.
        /// </summary>
        /// <param name="key">Key used to retrieve matching value</param>
        /// <param name="fields">List of fields for a work item</param>
        /// <returns></returns>
        public String RetrieveFieldValue(String key, IList<FieldInfo>
fields)
        {
            if (String.IsNullOrEmpty(key))
                return String.Empty;

            var result = fields.Single(s => s.Field.RefName == key);

            return result.Value;
        }

    }

    public class Content
    {
        public String SubscriptionId { get; set; }
```

```
        public int NotificationId { get; set; }

        public String EventType { get; set; }

        public WorkItemResource Resource { get; set; }

    }

    public class WorkItemResource
    {
        public String UpdatesUrl { get; set; }

        public IList<FieldInfo> Fields { get; set;}

        public int Id { get; set; }

        public int Rev { get; set; }

        public String Url { get; set; }

        public String WebUrl { get; set; }
    }

    public class FieldInfo
    {
        public FieldDetailedInfo Field { get; set; }

        public String Value { get; set; }

    }

    public class FieldDetailedInfo
    {
        public int Id { get; set; }

        public String Name { get; set; }

        public String RefName { get; set; }
    }
}
```

Related articles

- Authorize service hooks
- Consumers
- Events
- Troubleshooting and FAQs

Get started with the REST APIs

Article • 02/07/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Integrate your app with Azure DevOps using the REST APIs provided in this article.

The APIs follow a common pattern, like the following example.

```
no-highlight

VERB https://{instance}/{collection}/{team-project}/_apis/{area}/{resource}?
api-version={version}
```

> **Tip**
>
> As APIs evolve, we recommend that you include an API version in every request. This practice can help you avoid unexpected changes in the API that could break.

Azure DevOps Services

For Azure DevOps Services, `instance` is `dev.azure.com/{organization}` and `collection` is `DefaultCollection`, so the pattern looks like the following example.

```
no-highlight

VERB https://dev.azure.com/{organization}/_apis/{area}/{resource}?api-
version={version}
```

The following example shows how to get a list of projects in an organization.

```
dos

curl -u {username}:{personalaccesstoken}
https://dev.azure.com/{organization}/_apis/projects?api-version=2.0
```

If you want to provide the personal access token (PAT) through an HTTP header, you must first convert it to a Base64 string. The following example shows how to convert to Base64 using C#. The resulting string can then be provided as an HTTP header in the format.

```
Authorization: Basic BASE64PATSTRING
```

The following example shows C# using the HttpClient class.

cs

```cs
public static async void GetProjects()
{
    try
    {
        var personalaccesstoken = "PAT_FROM_WEBSITE";

        using (HttpClient client = new HttpClient())
        {
            client.DefaultRequestHeaders.Accept.Add(
                new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"))
;

            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic",
                Convert.ToBase64String(
                    System.Text.ASCIIEncoding.ASCII.GetBytes(
                        string.Format("{0}:{1}", "",
personalaccesstoken))));

            using (HttpResponseMessage response = client.GetAsync(

"https://dev.azure.com/{organization}/_apis/projects").Result)
            {
                response.EnsureSuccessStatusCode();
                string responseBody = await
response.Content.ReadAsStringAsync();
                Console.WriteLine(responseBody);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Most of our samples use PATs, as they're a compact example for authenticating with the service. However, there are various authentication mechanisms available for Azure DevOps Services including Microsoft Authentication Library (MSAL), OAuth, and Session

Tokens. For more information, see Authentication guidance, to help you determine which one is best suited for your scenario.

Azure DevOps Server

For Azure DevOps Server, `instance` is `{server:port}`. The default port for a non-SSL connection is 8080.

The default collection is `DefaultCollection`, but you can use any collection.

Here's how to get a list of projects from Azure DevOps Server using the default port and collection across SSL:

dos

```
curl -u {username}:{personalaccesstoken}
https://{server}/DefaultCollection/_apis/projects?api-version=2.0
```

To get the same list across a non-SSL connection:

dos

```
curl -u {username}:{personalaccesstoken}
http://{server}:8080/DefaultCollection/_apis/projects?api-version=2.0
```

These examples use PATs, which require that you create a PAT.

Responses

You should get a response like this.

JSON


```
{
    "value": [
        {
            "id": "eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "name": "Fabrikam-Fiber-TFVC",
            "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "description": "TeamFoundationVersionControlprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
```

```
                "collectionUrl": "https: //dev.azure.com/fabrikam-fiber-inc"
            },
            "defaultTeam": {
                "id": "66df9be7-3586-467b-9c5f-425b29afedfd",
                "name": "Fabrikam-Fiber-TFVCTeam",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1/teams/66df9be7-3586-
467b-9c5f-425b29afedfd"
            }
        },
        {
            "id": "6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "name": "Fabrikam-Fiber-Git",
            "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "description": "Gitprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "collectionUrl": "https: //dev.azure.com/fabrikam-fiber-inc"
            },
            "defaultTeam": {
                "id": "8bd35c5e-30bb-4834-a0c4-d576ce1b8df7",
                "name": "Fabrikam-Fiber-GitTeam",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c/teams/8bd35c5e-30bb-
4834-a0c4-d576ce1b8df7"
            }
        }
    ],
    "count": 2
}
```


The response is JSON, which is generally what you get back from the REST APIs, although there are a few exceptions, like Git blobs.

Now, you can look around the specific API areas like work item tracking or Git and get to the resources that you need. Keep reading to learn more about the general patterns that are used in these APIs.

HTTP verbs

Expand table

Verb	Used for...
GET	Get a resource or list of resources

Verb	Used for...
POST	Create a resource, Get a list of resources using a more advanced query
PUT	Create a resource if it doesn't exist or, if it does, update it
PATCH	Update a resource
DELETE	Delete a resource

Request headers and request content

When you provide request body (usually with the POST, PUT and PATCH verbs), include request headers that describe the body. For example,

no-highlight

```
POST https://dev.azure.com/fabrikam-fiber-inc/_apis/build-release/requests
```

HTTP

```
Content-Type: application/json
```

JSON

```
{
   "definition": {
      "id": 3
   },
   "reason": "Manual",
   "priority": "Normal"
}
```

HTTP method override

Some web proxies might only support the HTTP verbs GET and POST, but not more modern HTTP verbs like PATCH and DELETE. If your calls might pass through one of these proxies, you can send the actual verb using a POST method, with a header to override the method. For example, you might want to update a work item (`PATCH _apis/wit/workitems/3`), but you might have to go through a proxy that only allows GET or POST. You can pass the proper verb (PATCH in this case) as an HTTP request header parameter and use POST as the actual HTTP method.

no-highlight

```
POST https://dev.azure.com/fabrikam-fiber-inc/_apis/wit/workitems/3
```

HTTP

```
X-HTTP-Method-Override: PATCH
```

JSON

```
{
   (PATCH request body)
}
```

Response codes

Expand table

Response	Notes
200	Success, and there's a response body.
201	Success, when creating resources. Some APIs return 200 when successfully creating a resource. Look at the docs for the API you're using to be sure.
204	Success, and there's no response body. For example, you get this response when you delete a resource.
400	The parameters in the URL or in the request body aren't valid.
401	Authentication failed. Often, this response is because of a missing or malformed Authorization header.
403	The authenticated user doesn't have permission to do the operation.
404	The resource doesn't exist, or the authenticated user doesn't have permission to see that it exists.
409	There's a conflict between the request and the state of the data on the server. For example, if you attempt to submit a pull request and there's already a pull request for the commits, the response code is 409.

Cross-origin resource sharing (CORS)

Azure DevOps Services supports CORS, which enables JavaScript code served from a domain other than `dev.azure.com/*` to make Ajax requests to Azure DevOps Services

REST APIs. Each request must provide credentials (PATs and OAuth access tokens are both supported options). Example:

JavaScript

```
    $( document ).ready(function() {
        $.ajax({
            url: 'https://dev.azure.com/fabrikam/_apis/projects?api-
version=1.0',
            dataType: 'json',
            headers: {
                'Authorization': 'Basic ' + btoa("" + ":" + myPatToken)
            }
        }).done(function( results ) {
            console.log( results.value[0].id + " " + results.value[0].name
);
        });
    });
```

(replace `myPatToken` with a personal access token)

Versioning

Azure DevOps REST APIs are versioned to ensure applications and services continue to work as APIs evolve.

Guidelines

- Specify the API version with every request (**required**).
- Format API versions as follows: {major}.{minor}-{stage}.{resource-version}. For example, `1.0`, `1.1`, `1.2-preview`, `2.0`.
- Specify a particular revision of the API when it's in preview, by using the following version format: `1.0-preview.1`, `1.0-preview.2`. Once an API is released (1.0, for example), its preview version (1.0-preview) is deprecated and can be deactivated after 12 weeks.
- Upgrade to the released version of the API. Once a preview API is deactivated, requests that specify `-preview` version get rejected.

Usage

Specify the API version in the header of the HTTP request or as a URL query parameter.

HTTP request header:

HTTP

```
Accept: application/json;api-version=1.0
```

Query parameter:

no-highlight

```
GET https://dev.azure.com/{organization}/_apis/{area}/{resource}?api-version=1.0
```

Supported versions

For information on supported versions, see REST API versioning, Supported versions.

REST API versioning

Article • 12/18/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Using versioned REST APIs, you can ensure your projects are scalable and maintainable over time while applications and services continue to work as APIs evolve.

Guidelines

- API version **must** be specified with every request.
- API versions are in the format {major}.{minor}[-{stage}[.{resource-version}]] - For example, `1.0`, `1.1`, `1.2-preview`, `2.0`.
- While an API is in preview, you can specify a precise version of a particular revision of the API when needed (for example, `1.0-preview.1`, `1.0-preview.2`).
- Once an API is released (1.0, for example), its preview version (1.0-preview) is deprecated and can be deactivated after 12 weeks.
- During this time, you should upgrade to the released version of the API. Once a preview API is deactivated, requests that specify a `-preview` version get rejected.

Usage

The API version can be specified either in the header of the HTTP request or as a URL query parameter:

HTTP request header:

HTTP

```
Accept: application/json;api-version=1.0
```

Uri:

no-highlight

```
[scheme"://"][host[':'port]]"/v" major-version '/'namespace '/'resource
('/'resource)* '?' query

i.e. GET https://dev.azure.com/v1.0/{organization}/_apis/{area}/{resource}?
some-query=1000
```

Supported versions

> **Note**
>
> All supported Rest APIs are documented in the Azure DevOps REST API Reference

Expand table

Product	1.0	2.0	3.0	4.0	5.0	6.0	7.0
Azure DevOps Services	X	X	X	X	X	X	X
Azure DevOps Server 2022	X	X	X	X	X	X	X
Azure DevOps Server 2020	X	X	X	X	X	X	-
Azure DevOps Server 2019	X	X	X	X	X	-	-
Team Foundation Server 2018	X	X	X	X	-	-	-
Team Foundation Server 2017	X	X	X	-	-	-	-
Team Foundation Server 2015	X	X	-	-	-	-	-

Major API version releases align with Team Foundation Server RTM releases. For example, the `3.0` API set was introduced with Team Foundation Server 2017.

A few undocumented version 1.0 APIs existed in Team Foundation Server 2013, but aren't supported.

Related articles

- Get started with REST APIs
- Azure DevOps Services REST API reference

Feedback

Was this page helpful? Yes No

Provide product feedback

.NET client libraries

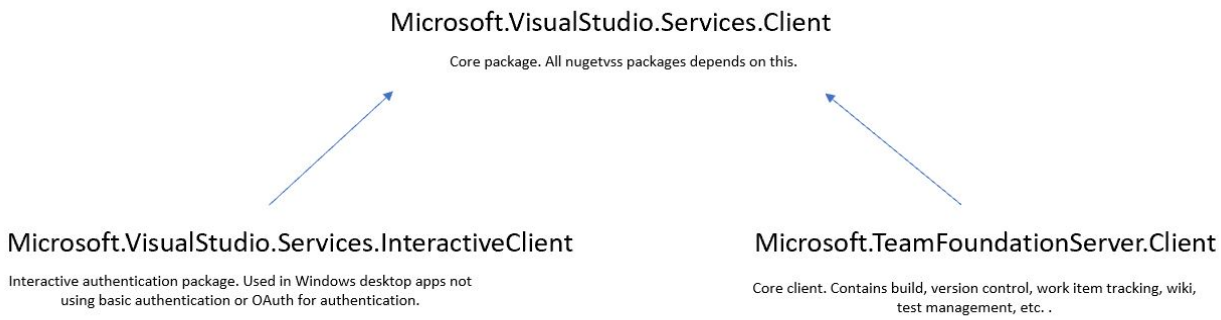
Article • 11/10/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Client libraries are available for .NET developers who build Windows apps and services that integrate with Azure DevOps. Client libraries integrate with work item tracking, version control, build, and other services. These packages replace the traditional TFS Client OM installer and make it easy to acquire and redistribute the libraries needed by your app or service.

> **Tip**
>
> Find the corresponding .NET client class and API method, using **.NET Client Library Samples** and **Azure DevOps API Reference**.

Dependency diagram



Features

- Downloadable from NuGet.org and easily importable into your Visual Studio projects
- Libraries are licensed for redistribution in your apps and services (view the license)
- Access both traditional client object model APIs and REST APIs

> **Note**
>
> REST-based clients only work with Azure DevOps and TFS 2015 or later. For more information about extending and integrating with Azure DevOps using the client libraries, see **Extending Team Foundation**.

Package and Azure DevOps version-mapping table

Package version	Azure DevOps version
16.205.x	`versions >= Azure DevOps Server 2022`
16.170.x	`versions >= Azure DevOps Server 2020`
16.153.x	`versions >= Azure DevOps Server 2019 Update 1`
16.143.x	`versions >= Azure DevOps Server 2019`
15.131.x	`versions >= TFS 2018 Update 1`
15.112.x	`versions >= TFS 2017 Update 1`
14.102.x	`versions >= TFS 2015 Update 3`

Package version	Azure DevOps version
14.95.x	`versions >= TFS 2015 Update 2`
14.89.x	`versions >= TFS 2015 Update 1`
14.83.x	`versions >= TFS 2015`

For the latest preview versions, see the NuGet packages gallery.

REST packages

Packages and description	Binaries
Microsoft.VisualStudio.Services.Client Provides access to shared platform services via public REST APIs, like organization, profile, identity, security, and more.	`Microsoft.VisualStudio.Services.WebApi.dll`, `Microsoft.VisualStudio.Services.Common.dll`, `Microsoft.TeamFoundation.Common.dll`
Microsoft.TeamFoundationServer.Client Provides access to version control, work item tracking, build, and more via public REST APIs.	`Microsoft.TeamFoundation.Build2.WebApi.dll`, `Microsoft.TeamFoundation.Core.WebApi.dll`, `Microsoft.TeamFoundation.WorkItemTracking.Process.WebApi.dll`, `Microsoft.TeamFoundation.SourceControl.WebApi.dll`, `Microsoft.TeamFoundation.TestManagement.WebApi.dll`, and so on.
Microsoft.VisualStudio.Services.InteractiveClient Supports applications that require interactive sign-in by a user.	`Microsoft.VisualStudio.Services.Client.Interactive.dll`
Microsoft.VisualStudio.Services.Release.Client Provides access to the Release Service via public REST APIs.	`Microsoft.VisualStudio.Services.ReleaseManagement.WebApi.dll`
Microsoft.TeamFoundation.DistributedTask.Common.Contracts Provides the models used to access the Distributed Task Service via public REST APIs.	`Microsoft.TeamFoundation.DistributedTask.Common.Contracts.dll`
Microsoft.TeamFoundation.DistributedTask.WebApi Provides access to the Distributed Task Service via public REST APIs.	`Microsoft.TeamFoundation.DistributedTask.WebApi.dll`
Microsoft.VisualStudio.Services.ServiceHooks.WebApi Provides access to the Service Hooks Service via public REST APIs.	`Microsoft.VisualStudio.Services.ServiceHooks.WebApi.dll`
Microsoft.VisualStudio.Services.Gallery.WebApi Provides access to the Gallery Service via public REST APIs.	`Microsoft.VisualStudio.Services.Gallery.WebApi.dll`
Microsoft.VisualStudio.Services.Notifications.WebApi Provides access to the Notifications Service via public REST APIs.	`Microsoft.VisualStudio.Services.Notifications.WebApi.dll`
Microsoft.VisualStudio.Services.ExtensionManagement.WebApi Provides access to the Extension Management Service via public REST APIs.	`Microsoft.VisualStudio.Services.ExtensionManagement.WebApi.dll`
Microsoft.VisualStudio.Services.MemberEntitlementManagement.WebApi Provides access to the Member Entitlement Management Service via public REST APIs.	`Microsoft.VisualStudio.Services.MemberEntitlementManagement.WebApi.dll`
Microsoft.VisualStudio.Services.ServiceEndpoints.WebApi Provides access to the Service Endpoints via public REST APIs.	`Microsoft.VisualStudio.Services.ServiceEndpoints.WebApi.dll`
Microsoft.VisualStudio.Services.Search.Client Provides access to the Search Service via public REST APIs.	`Microsoft.VisualStudio.Services.Search.Shared.WebApi.dll`, `Microsoft.VisualStudio.Services.Search.WebApi.dll`
Microsoft.TeamFoundation.PublishTestResults This task can be used to Publish test results and upload test attachments on Azure DevOps. The following results formats are supported with this package: 1. JUnit - publish tests from Junit projects, 2. NUnit- publish tests from Nunit projects, 3. VSTest- publish tests from Visual Studio projects, 4. Xunit- publish tests from Xunit projects	`Microsoft.TeamFoundation.TestClient.PublishTestResults.dll`
Microsoft.VisualStudio.Services.Audit.WebApi Provides access to the Audit Service via public REST APIs.	`Microsoft.VisualStudio.Services.Audit.WebApi.dll`

Packages and description	Binaries
Microsoft.Azure.Pipelines.Policy.Client Provides access to the pipeline approvals, checks, and authorization via public REST APIs.	`Microsoft.Azure.Pipelines.Policy.Client.dll`

> **Tip**
>
> If you have an existing Windows application or service that uses the TFS Client Object Model, use `Microsoft.TeamFoundationServer.ExtendedClient`.

Soap package

Package and description	Binaries
Microsoft.TeamFoundationServer.ExtendedClient Work with and manage version control, work items, and build, and other resources from your client application. This package doesn't support Net Standard Client OM. Only use this package when our REST APIs don't offer the functionality you need (for example, creating workspaces in TFVC)	`Microsoft.TeamFoundation.Build.Client.dll`, `Microsoft.TeamFoundation.DeleteTeamProject.dll`, `Microsoft.TeamFoundation.Diff.dll`, `Microsoft.TeamFoundation.Git.Client.dll`, `Microsoft.TeamFoundation.SharePointReporting.Integration.dll`, and so on.

Install

Install from a NuGet package manager command prompt:

PowerShell

```
PM> Install-Package Microsoft.TeamFoundationServer.ExtendedClient
```

Connect

To create an authenticated connection to Azure DevOps, get an HttpClient for the service you want to work with, and then call methods against that service.

See the following examples:

C#

```
using Microsoft.VisualStudio.Services.Common;
using Microsoft.VisualStudio.Services.Client;
using Microsoft.TeamFoundation.SourceControl.WebApi;
using Microsoft.VisualStudio.Services.WebApi;

const String c_collectionUri = "https://dev.azure.com/fabrikam";
const String c_projectName = "MyGreatProject";
const String c_repoName = "MyRepo";

Uri orgUrl = new Uri(c_collectionUri);

// Connect to Azure DevOps Services
VssConnection connection = new VssConnection(orgUrl, new VssBasicCredential(string.Empty, personalAccessToken));

// Get a GitHttpClient to talk to the Git endpoints
using (GitHttpClient gitClient = connection.GetClient<GitHttpClient>())
{
    // Get data about a specific repository
    var repo = gitClient.GetRepositoryAsync(c_projectName, c_repoName).Result;
}
```

Authentication paths that produce an interactive dialog aren't available in the .NET Standard version of the .NET client libraries. When you're using the .NET Standard version, provide your credentials more explicitly to authenticate, like in the following

example.

```C#
using System;
using Microsoft.TeamFoundation.SourceControl.WebApi;
using Microsoft.VisualStudio.Services.Common;
using Microsoft.VisualStudio.Services.WebApi;

namespace ConsoleApp1
{
    class Program
    {
        const string collectionUri = "https://dev.azure.com/fabrikam";
        const string projectName = "MyGreatProject";
        const string repoName = "MyRepo";
        const string pat = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";

        static void Main(string[] args)
        {
            var creds = new VssBasicCredential(string.Empty, pat);

            // Connect to Azure DevOps Services
            var connection = new VssConnection(new Uri(collectionUri), creds);

            // Get a GitHttpClient to talk to the Git endpoints
            using var gitClient = connection.GetClient<GitHttpClient>();

            // Get data about a specific repository
            var repo = gitClient.GetRepositoryAsync(projectName, repoName).Result;
        }
    }
}
```

For more authentication samples, see .NET Samples.

Reference

For more information and for up-to-date reference documentation, see the .NET API browser.

Samples

Check out samples on our .NET Samples Page or directly on our .NET GitHub Repo.

For more information on NuGet packages, see Microsoft.VisualStudio.Services.Client.

Known issues

Interactive authentication dialog doesn't appear when using the Azure DevOps OM in a Single Threaded Apartment (STA)

The interactive authentication dialog doesn't appear in cases where your code is running from a Single Threaded Apartment (STA). This issue can commonly occur from WPF applications. To work around this issue, you can change your initialization method to be async and request authentication as in the following example.

```C#
async void InitAzureDevOps()
{
    Uri _uri = new Uri("https://dev.azure.com/MyAccount/");

    var creds = new VssClientCredentials(new WindowsCredential(false),
                                         new VssFederatedCredential(false),
                                         CredentialPromptType.PromptIfNeeded);
```

```
    VssConnection vssConnection = new VssConnection(_uri, creds);
    await vssConnection.ConnectAsync();

    ...

}
```

Using NetStandard 2.0 versions of the Azure DevOps OM

For version 16.143.1 of our NuGet packages, we support NetStandard 2.0. These packages correlate with Azure DevOps Server 2019 RTW and are fully compatible with Azure DevOps.

Microsoft.TeamFoundationServer.ExtendedClient package doesn't have NetStandard support

The Microsoft.TeamFoundationServer.ExtendedClient doesn't support a NetStandard compliant version.

> **Important**
>
> This package includes our older SOAP object model, which we replaced by our newer REST object model. We're no longer investing in the older SOAP object model, and have no plans to create a NetStandard version of it.

Get started with the REST APIs

Article • 02/07/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Integrate your app with Azure DevOps using the REST APIs provided in this article.

The APIs follow a common pattern, like the following example.

```
no-highlight

VERB https://{instance}/{collection}/{team-project}/_apis/{area}/{resource}?
api-version={version}
```

> **Tip**
>
> As APIs evolve, we recommend that you include an API version in every request. This practice can help you avoid unexpected changes in the API that could break.

Azure DevOps Services

For Azure DevOps Services, `instance` is `dev.azure.com/{organization}` and `collection` is `DefaultCollection`, so the pattern looks like the following example.

```
no-highlight

VERB https://dev.azure.com/{organization}/_apis/{area}/{resource}?api-
version={version}
```

The following example shows how to get a list of projects in an organization.

```
dos

curl -u {username}:{personalaccesstoken}
https://dev.azure.com/{organization}/_apis/projects?api-version=2.0
```

If you want to provide the personal access token (PAT) through an HTTP header, you must first convert it to a Base64 string. The following example shows how to convert to Base64 using C#. The resulting string can then be provided as an HTTP header in the format.

```
Authorization: Basic BASE64PATSTRING
```

The following example shows C# using the HttpClient class.

cs

```cs
public static async void GetProjects()
{
    try
    {
        var personalaccesstoken = "PAT_FROM_WEBSITE";

        using (HttpClient client = new HttpClient())
        {
            client.DefaultRequestHeaders.Accept.Add(
                new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"))
;

            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic",
                Convert.ToBase64String(
                    System.Text.ASCIIEncoding.ASCII.GetBytes(
                        string.Format("{0}:{1}", "",
personalaccesstoken))));

            using (HttpResponseMessage response = client.GetAsync(

"https://dev.azure.com/{organization}/_apis/projects").Result)
            {
                response.EnsureSuccessStatusCode();
                string responseBody = await
response.Content.ReadAsStringAsync();
                Console.WriteLine(responseBody);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Most of our samples use PATs, as they're a compact example for authenticating with the service. However, there are various authentication mechanisms available for Azure DevOps Services including Microsoft Authentication Library (MSAL), OAuth, and Session

Tokens. For more information, see Authentication guidance, to help you determine which one is best suited for your scenario.

Azure DevOps Server

For Azure DevOps Server, `instance` is `{server:port}`. The default port for a non-SSL connection is 8080.

The default collection is `DefaultCollection`, but you can use any collection.

Here's how to get a list of projects from Azure DevOps Server using the default port and collection across SSL:

dos

```
curl -u {username}:{personalaccesstoken}
https://{server}/DefaultCollection/_apis/projects?api-version=2.0
```

To get the same list across a non-SSL connection:

dos

```
curl -u {username}:{personalaccesstoken}
http://{server}:8080/DefaultCollection/_apis/projects?api-version=2.0
```

These examples use PATs, which require that you create a PAT.

Responses

You should get a response like this.

JSON


```
{
    "value": [
        {
            "id": "eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "name": "Fabrikam-Fiber-TFVC",
            "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "description": "TeamFoundationVersionControlprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
```

```
                "collectionUrl": "https: //dev.azure.com/fabrikam-fiber-inc"
            },
            "defaultTeam": {
                "id": "66df9be7-3586-467b-9c5f-425b29afedfd",
                "name": "Fabrikam-Fiber-TFVCTeam",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1/teams/66df9be7-3586-
467b-9c5f-425b29afedfd"
            }
        },
        {
            "id": "6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "name": "Fabrikam-Fiber-Git",
            "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "description": "Gitprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "collectionUrl": "https: //dev.azure.com/fabrikam-fiber-inc"
            },
            "defaultTeam": {
                "id": "8bd35c5e-30bb-4834-a0c4-d576ce1b8df7",
                "name": "Fabrikam-Fiber-GitTeam",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c/teams/8bd35c5e-30bb-
4834-a0c4-d576ce1b8df7"
            }
        }
    ],
    "count": 2
}
```


The response is JSON, which is generally what you get back from the REST APIs, although there are a few exceptions, like Git blobs.

Now, you can look around the specific API areas like work item tracking or Git and get to the resources that you need. Keep reading to learn more about the general patterns that are used in these APIs.

HTTP verbs

Expand table

Verb	Used for...
GET	Get a resource or list of resources

Verb	Used for...
POST	Create a resource, Get a list of resources using a more advanced query
PUT	Create a resource if it doesn't exist or, if it does, update it
PATCH	Update a resource
DELETE	Delete a resource

Request headers and request content

When you provide request body (usually with the POST, PUT and PATCH verbs), include request headers that describe the body. For example,

no-highlight

```
POST https://dev.azure.com/fabrikam-fiber-inc/_apis/build-release/requests
```

HTTP

```
Content-Type: application/json
```

JSON

```
{
   "definition": {
      "id": 3
   },
   "reason": "Manual",
   "priority": "Normal"
}
```

HTTP method override

Some web proxies might only support the HTTP verbs GET and POST, but not more modern HTTP verbs like PATCH and DELETE. If your calls might pass through one of these proxies, you can send the actual verb using a POST method, with a header to override the method. For example, you might want to update a work item (`PATCH _apis/wit/workitems/3`), but you might have to go through a proxy that only allows GET or POST. You can pass the proper verb (PATCH in this case) as an HTTP request header parameter and use POST as the actual HTTP method.

no-highlight

```
POST https://dev.azure.com/fabrikam-fiber-inc/_apis/wit/workitems/3
```

HTTP

```
X-HTTP-Method-Override: PATCH
```

JSON

```
{
   (PATCH request body)
}
```

Response codes

Expand table

Response	Notes
200	Success, and there's a response body.
201	Success, when creating resources. Some APIs return 200 when successfully creating a resource. Look at the docs for the API you're using to be sure.
204	Success, and there's no response body. For example, you get this response when you delete a resource.
400	The parameters in the URL or in the request body aren't valid.
401	Authentication failed. Often, this response is because of a missing or malformed Authorization header.
403	The authenticated user doesn't have permission to do the operation.
404	The resource doesn't exist, or the authenticated user doesn't have permission to see that it exists.
409	There's a conflict between the request and the state of the data on the server. For example, if you attempt to submit a pull request and there's already a pull request for the commits, the response code is 409.

Cross-origin resource sharing (CORS)

Azure DevOps Services supports CORS, which enables JavaScript code served from a domain other than `dev.azure.com/*` to make Ajax requests to Azure DevOps Services

REST APIs. Each request must provide credentials (PATs and OAuth access tokens are both supported options). Example:

JavaScript

```
    $( document ).ready(function() {
        $.ajax({
            url: 'https://dev.azure.com/fabrikam/_apis/projects?api-
version=1.0',
            dataType: 'json',
            headers: {
                'Authorization': 'Basic ' + btoa("" + ":" + myPatToken)
            }
        }).done(function( results ) {
            console.log( results.value[0].id + " " + results.value[0].name
);
        });
    });
```

(replace `myPatToken` with a personal access token)

Versioning

Azure DevOps REST APIs are versioned to ensure applications and services continue to work as APIs evolve.

Guidelines

- Specify the API version with every request (**required**).
- Format API versions as follows: {major}.{minor}-{stage}.{resource-version}. For example, `1.0`, `1.1`, `1.2-preview`, `2.0`.
- Specify a particular revision of the API when it's in preview, by using the following version format: `1.0-preview.1`, `1.0-preview.2`. Once an API is released (1.0, for example), its preview version (1.0-preview) is deprecated and can be deactivated after 12 weeks.
- Upgrade to the released version of the API. Once a preview API is deactivated, requests that specify `-preview` version get rejected.

Usage

Specify the API version in the header of the HTTP request or as a URL query parameter.

HTTP request header:

HTTP

```
Accept: application/json;api-version=1.0
```

Query parameter:

no-highlight

```
GET https://dev.azure.com/{organization}/_apis/{area}/{resource}?api-
version=1.0
```

Supported versions

For information on supported versions, see REST API versioning, Supported versions.

C# client library samples

Article • 07/02/2024

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

The following samples show you how to extend and integrate with Azure DevOps using the .NET client libraries.

Samples in GitHub

On the .NET Sample GitHub page, you can find many samples with instructions on how to run them.

Other samples

REST examples on this page require the following NuGet packages:

- Microsoft.TeamFoundationServer.Client
- Microsoft.VisualStudio.Services.Client
- Microsoft.VisualStudio.Services.InteractiveClient

Example: Using a REST-based HTTP client

cs

```
// https://www.nuget.org/packages/Microsoft.TeamFoundationServer.Client/
using Microsoft.TeamFoundation.WorkItemTracking.WebApi;
using Microsoft.TeamFoundation.WorkItemTracking.WebApi.Models;

//
https://www.nuget.org/packages/Microsoft.VisualStudio.Services.InteractiveCl
ient/
using Microsoft.VisualStudio.Services.Client;

// https://www.nuget.org/packages/Microsoft.VisualStudio.Services.Client/
using Microsoft.VisualStudio.Services.Common;

/// <summary>
/// This sample creates a new work item query for New Bugs, stores it under
'MyQueries', runs the query, and then sends the results to the console.
/// </summary>
public static void SampleREST()
{
    // Connection object could be created once per application and we use it
to get httpclient objects.
```

```
    // Httpclients have been reused between callers and threads.
    // Their lifetime has been managed by connection (we don't have to
dispose them).
    // This is more robust then newing up httpclient objects directly.

    // Be sure to send in the full collection uri, i.e.
http://myserver:8080/tfs/defaultcollection
    // We are using default VssCredentials which uses NTLM against an Azure
DevOps Server.  See additional provided
    // Create a connection with PAT for authentication
    VssConnection connection = new VssConnection(orgUrl, new
VssBasicCredential(string.Empty, personalAccessToken));


    // Create instance of WorkItemTrackingHttpClient using VssConnection
    WorkItemTrackingHttpClient witClient =
connection.GetClient<WorkItemTrackingHttpClient>();

    // Get 2 levels of query hierarchy items
    List<QueryHierarchyItem> queryHierarchyItems =
witClient.GetQueriesAsync(teamProjectName, depth: 2).Result;

    // Search for 'My Queries' folder
    QueryHierarchyItem myQueriesFolder =
queryHierarchyItems.FirstOrDefault(qhi => qhi.Name.Equals("My Queries"));
    if (myQueriesFolder != null)
    {
        string queryName = "REST Sample";

        // See if our 'REST Sample' query already exists under 'My Queries'
folder.
        QueryHierarchyItem newBugsQuery = null;
        if (myQueriesFolder.Children != null)
        {
            newBugsQuery = myQueriesFolder.Children.FirstOrDefault(qhi =>
qhi.Name.Equals(queryName));
        }
        if (newBugsQuery == null)
        {
            // if the 'REST Sample' query does not exist, create it.
            newBugsQuery = new QueryHierarchyItem()
            {
                Name = queryName,
                Wiql = "SELECT [System.Id],[System.WorkItemType],
[System.Title],[System.AssignedTo],[System.State],[System.Tags] FROM
WorkItems WHERE [System.TeamProject] = @project AND [System.WorkItemType] =
'Bug' AND [System.State] = 'New'",
                IsFolder = false
            };
            newBugsQuery = witClient.CreateQueryAsync(newBugsQuery,
teamProjectName, myQueriesFolder.Name).Result;
        }

        // run the 'REST Sample' query
        WorkItemQueryResult result =
```

```
witClient.QueryByIdAsync(newBugsQuery.Id).Result;

        if (result.WorkItems.Any())
        {
            int skip = 0;
            const int batchSize = 100;
            IEnumerable<WorkItemReference> workItemRefs;
            do
            {
                workItemRefs = result.WorkItems.Skip(skip).Take(batchSize);
                if (workItemRefs.Any())
                {
                    // get details for each work item in the batch
                    List<WorkItem> workItems =
witClient.GetWorkItemsAsync(workItemRefs.Select(wir => wir.Id)).Result;
                    foreach (WorkItem workItem in workItems)
                    {
                        // write work item to console
                        Console.WriteLine("{0} {1}", workItem.Id,
workItem.Fields["System.Title"]);
                    }
                }
                skip += batchSize;
            }
            while (workItemRefs.Count() == batchSize);
        }
        else
        {
            Console.WriteLine("No work items were returned from query.");
        }
    }
}
```

Authentication

To change the method of authentication for Azure DevOps, change the VssCredential type passed to VssConnection when you create it.

Personal access token authentication for REST services

cs

```
public static void PersonalAccessTokenRestSample()
{
    // Create instance of VssConnection using Personal Access Token
    VssConnection connection = new VssConnection(orgUrl, new
VssBasicCredential(string.Empty, personalAccessToken));
}
```

Visual Studio sign-in prompt (Microsoft Account or Microsoft Entra backend) for REST services (.NET Framework only)

Since the .NET Core version doesn't support interactive dialogs, this sample applies only to the .NET Framework version of the clients.

cs

```cs
public static void MicrosoftAccountRestSample()
{
    // Create instance of VssConnection using Visual Studio sign-in prompt
    VssConnection connection = new VssConnection(new Uri(collectionUri), new
VssClientCredentials());
}
```

Microsoft Entra authentication for REST services

cs

```cs
public static void AADRestSample()
{
    // Create instance of VssConnection using Azure AD Credentials for Azure
AD backed account
    VssConnection connection = new VssConnection(new Uri(collectionUri), new
VssAadCredential(userName, password));
}
```

OAuth Authentication for REST services

For more information, see Azure DevOps auth samples and Microsoft identity platform and OAuth 2.0 authorization code flow.

cs

```cs
public static void OAuthSample()
{
    // Create instance of VssConnection using OAuth Access token
    VssConnection connection = new VssConnection(new Uri(collectionUri), new
VssOAuthAccessTokenCredential(accessToken));
}
```

Feedback

Was this page helpful? Yes No

Provide product feedback

C# client library samples for SOAP clients and services

Article • 11/10/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

This article contains samples that show you how to extend and integrate with Azure DevOps Server and Azure DevOps Services using the legacy SOAP clients. These clients are only available in the .NET Framework version of the clients.

> **ⓘ Important**
>
> For new development, see the JSON-based clients described in **.NET client libraries**.

Prerequisites

Examples on this page require the following NuGet packages:

- Microsoft.TeamFoundationServer.ExtendedClient
- Microsoft.TeamFoundationServer.Client
- Microsoft.VisualStudio.Services.Client
- Microsoft.VisualStudio.Services.InteractiveClient

Example: Using SOAP-based client

C#

```
//
https://www.nuget.org/packages/Microsoft.TeamFoundationServer.ExtendedClient
/
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.WorkItemTracking.Client;

/// <summary>
/// This sample creates a new work item query under 'MyQueries', runs the
query, and then sends the results to the console.
/// </summary>
public static void SampleSOAP()
{
    // create TfsTeamProjectCollection instance using default credentials
```

```
    using (TfsTeamProjectCollection tpc = new TfsTeamProjectCollection(new
Uri(collectionUri)))
    {
        // get the WorkItemStore service
        WorkItemStore workItemStore = tpc.GetService<WorkItemStore>();

        // get the project context for the work item store
        Project workItemProject = workItemStore.Projects[teamProjectName];

        // search for the 'My Queries' folder
        QueryFolder myQueriesFolder =
workItemProject.QueryHierarchy.FirstOrDefault(qh => qh is QueryFolder &&
qh.IsPersonal) as QueryFolder;
        if (myQueriesFolder != null)
        {
            // search for the 'SOAP Sample' query
            string queryName = "SOAP Sample";
            QueryDefinition newBugsQuery = myQueriesFolder.FirstOrDefault(qi
=> qi is QueryDefinition && qi.Name.Equals(queryName)) as QueryDefinition;
            if (newBugsQuery == null)
            {
                // if the 'SOAP Sample' query does not exist, create it.
                newBugsQuery = new QueryDefinition(queryName, "SELECT
[System.Id],[System.WorkItemType],[System.Title],[System.AssignedTo],
[System.State],[System.Tags] FROM WorkItems WHERE [System.WorkItemType] =
'Bug' AND [System.State] = 'New'");
                myQueriesFolder.Add(newBugsQuery);
                workItemProject.QueryHierarchy.Save();
            }

            // run the 'SOAP Sample' query
            WorkItemCollection workItems =
workItemStore.Query(newBugsQuery.QueryText);
            foreach (WorkItem workItem in workItems)
            {
                // write work item to console
                Console.WriteLine("{0} {1}", workItem.Id,
workItem.Fields["System.Title"].Value);
            }
        }
    }
}
```

Authentication

To change the method of authentication to Azure DevOps, change the VssCredential type passed to VssConnection when you create it.

Personal access token authentication for SOAP services

C#

```
public static void PersonalAccessTokenSoapSample()
{
    // Authenticate using Personal Access Token
    VssBasicCredential vssBasicCredential = new
VssBasicCredential(string.Empty, pat);
    using (TfsTeamProjectCollection tpc = new TfsTeamProjectCollection(new
Uri(collectionUri), vssBasicCredential))
    {
        tpc.Authenticate();
        Console.WriteLine(tpc.InstanceId);
    }
}
```

Microsoft Entra authentication for SOAP services

C#

```
public static void AADSoapSample()
{
    // Authenticate using Azure Active Directory credential (requires a
Azure AD-backed organization)
    using (TfsTeamProjectCollection tpc = new TfsTeamProjectCollection(new
Uri(collectionUri), new VssAadCredential()))
    {
        tpc.Authenticate();
        Console.WriteLine(tpc.InstanceId);
    }
}
```

Visual Studio sign-in prompt (Microsoft Account or Microsoft Entra backed) for SOAP services

C#

```
public static void MicrosoftAccountSample()
{
    // authenticate using Visual Studio sign-in prompt
    using (TfsTeamProjectCollection tpc = new TfsTeamProjectCollection(new
Uri(collectionUri), new VssClientCredentials()))
    {
        tpc.Authenticate();
        Console.WriteLine(tpc.InstanceId);
    }
}
```

REST API samples for Azure DevOps

Article • 11/10/2023

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

Most samples in this article use personal access tokens (PATs). PATs are a compact example for authentication. There are many other authentication mechanisms available, including Microsoft Authentication Library, OAuth, and Session tokens. For more information to gauge which is best suited for your scenario, see Authentication guidance.

For more information, see Azure DevOps Services REST API Reference and Get started with REST APIs.

Personal access tokens

Authenticate with Azure DevOps when you use the REST APIs or .NET Libraries.

Get started with these samples and create a PAT.

> **Tip**
>
> PATs are like passwords. Keep them secret. Make sure you save them in a secure location once your PAT gets created.

To provide the PAT through an HTTP header, first convert it to a Base64 string. The following example shows how to convert to Base64 using C#. You can provide the resulting string as an HTTP header in the following format:

```
Authorization: Basic BASE64USERNAME:PATSTRING
```

REST API

See the following example of getting a list of projects for your organization via REST API.

C#

```
using System.Net.Http;
using System.Net.Http.Headers;

...
```

```
//encode your personal access token
string credentials =
Convert.ToBase64String(System.Text.ASCIIEncoding.ASCII.GetBytes(string.Forma
t("{0}:{1}", "", personalAccessToken)));

ListOfProjectsResponse.Projects viewModel = null;

//use the httpclient
using (var client = new HttpClient())
{
    client.BaseAddress = new Uri($"https://dev.azure.com/{OrgName}/");
//url of your organization
    client.DefaultRequestHeaders.Accept.Clear();
    client.DefaultRequestHeaders.Accept.Add(new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"))
;
    client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic", credentials);

    //connect to the REST endpoint
    HttpResponseMessage response = client.GetAsync("_apis/projects?
stateFilter=All&api-version=1.0").Result;

    //check to see if we have a successful response
    if (response.IsSuccessStatusCode)
    {
        //set the viewmodel from the content in the response
        viewModel =
response.Content.ReadAsAsync<ListOfProjectsResponse.Projects>().Result;

        //var value = response.Content.ReadAsStringAsync().Result;
    }
}
```

.NET Client Libraries

Here, we're using two of the .NET Client Libraries. Make sure you reference the following .NET Client Libraries within your .NET project.

- Microsoft.TeamFoundationServer.Client
- Microsoft.Visual Studio.Services.Client

See the following example of getting a list of projects for your organization via .NET Client Libraries.

```
C#
```

```
using Microsoft.TeamFoundation.Core.WebApi;
using Microsoft.VisualStudio.Services.Common;

...

//create uri and VssBasicCredential variables
Uri uri = new Uri(url);
VssBasicCredential credentials = new VssBasicCredential("",
personalAccessToken);

using (ProjectHttpClient projectHttpClient = new ProjectHttpClient(uri,
credentials))
{
    IEnumerable<TeamProjectReference> projects =
projectHttpClient.GetProjects().Result;
}
```

FAQs

Q: Where can I get the source code for the code samples?

A: See the https://github.com/Microsoft/vsts-restapi-samplecode.

Q: Where can I find more information on the .NET library?

A: See the overview of client libraries.

Create a pull request status server with Node.js

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The pull request (PR) workflow provides developers with an opportunity to get feedback on their code from peers as well as from automated tools. 3rd party tools and services can participate in the PR workflow by using the PR Status API. This article guides you through the process of creating a status server to validate PRs in an Azure DevOps Services Git repository. For more information about PR status, see Customize and extend pull request workflows with pull request status.

Prerequisites

- An organization in Azure DevOps with a Git repo. If you don't have an organization, sign up to upload and share code in free unlimited private Git repositories.
- Install VS Code or other code editor of your choice. The instructions in this guide use VS Code but the steps in other code editors are similar.

Install Node.js

To install Node.js, download the LTS release appropriate for your platform. The download contains an installer, which you can run to install the Node.js runtime on your local machine. When installing Node.js, be sure to keep the npm package manager portion of the install, which is selected by default.

Create a basic web server using Express

The steps in this section use Express, which is a lightweight web framework for Node.js that provides a number of HTTP utility methods that simplify creating a web server. This framework provides you with the basic functions needed to listen to PR events.

1. From the command line, create a new project folder for your web server.

```
```

```
mkdir pr-server
cd pr-server
```

2. Use the `npm init` command to create a new `package.json` file for the project.

```

npm init
```

   Press Enter to accept the defaults for all of the options except the entry point. Change it to `app.js`

```

entry point: (index.js) app.js
```

3. Install Express in the pr-server directory using the following command. This installs Express and saves it to the dependencies list.

```

npm install express
```

4. Create a simple Express app to build upon for the PR status server. The following steps are based on the Express Hello world example. Open the project folder in VS Code by running the following command from the `pr-server` folder.

```

code .
```

5. Create a new file `(Ctrl + N)` and paste in the following sample code.

JavaScript

```
const express = require('express')
const app = express()

app.get('/', function (req, res) {
res.send('Hello World!')
})

app.listen(3000, function () {
```

```
console.log('Example app listening on port 3000!')
})
```

6. Save the file as `app.js`.

7. Run the basic web server using the following command:

```
node app.js
```

   Verify the server is running by browsing to `http://localhost:3000/`.

Listen for HTTP POST requests

The web server is going to receive `POST` requests from Azure DevOps Services, so you need to handle those requests in your server.

1. At the end of the `app.js` file, add the following code, and save the file.

```
JavaScript

app.post('/', function (req, res) {
    res.send('Received the POST')
})
```

2. Re-run your web server using the following command:

```
node app.js
```

Configure a service hook for PR events

Service hooks are an Azure DevOps Services feature that can alert external services when certain events occur. For this sample, you'll want to set up two service hooks for PR events, so the status server can be notified. The first will be for the **Pull request created** event and the second will be for the **Pull request updated** event.

In order to receive the service hook notifications, you'll need to expose a port to the public internet. The ngrok utility is very useful for doing this in a development environment.

1. Download and unzip the appropriate ngrok release for your platform.

2. Use ngrok to start listening on the same port as your sample server - port 3000. Run the following command in a new command window.

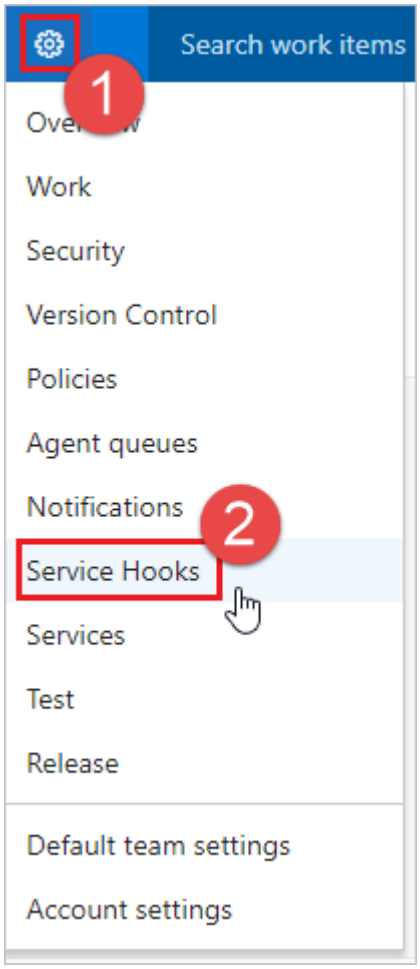
```
ngrok http 3000
```

   Ngrok will create a public URL that forwards to `localhost:3000`. Note that URL as you will need it in the next step. It will look something like this:

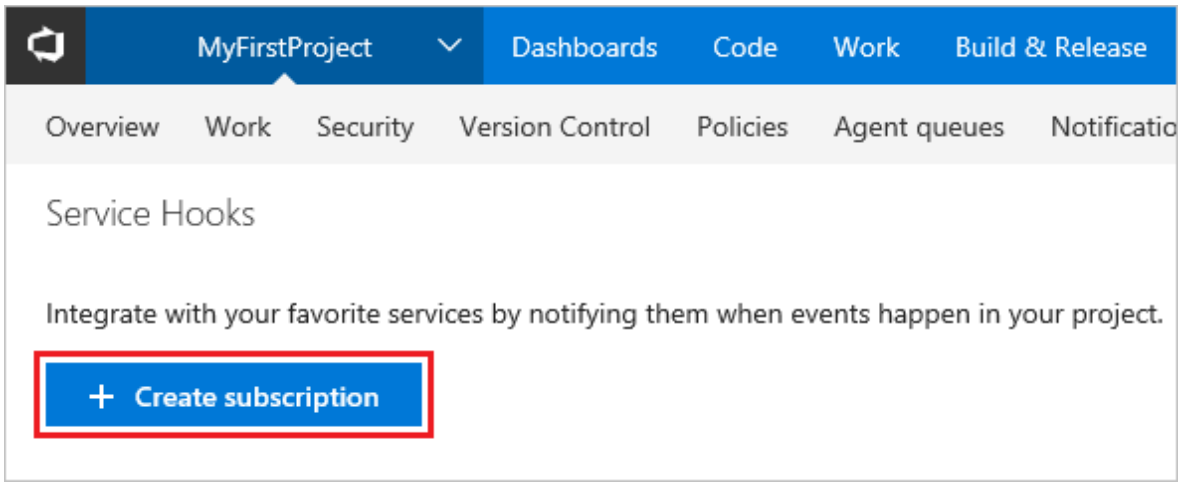
```
http://c3c1bffa.ngrok.io
```

3. Browse to your project in Azure DevOps, e.g. `https://dev.azure.com/<your account>/<your project name>`

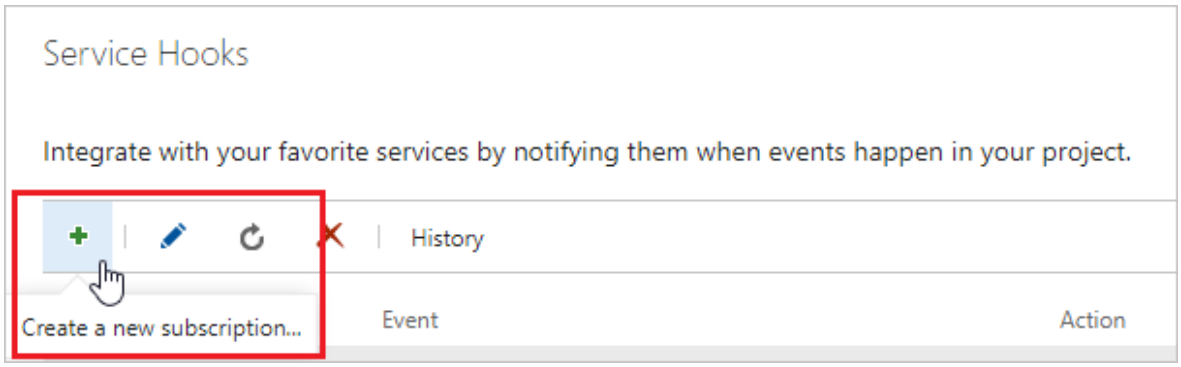
4. From the navigation menu, hover over the **gear** and select **Service Hooks**.



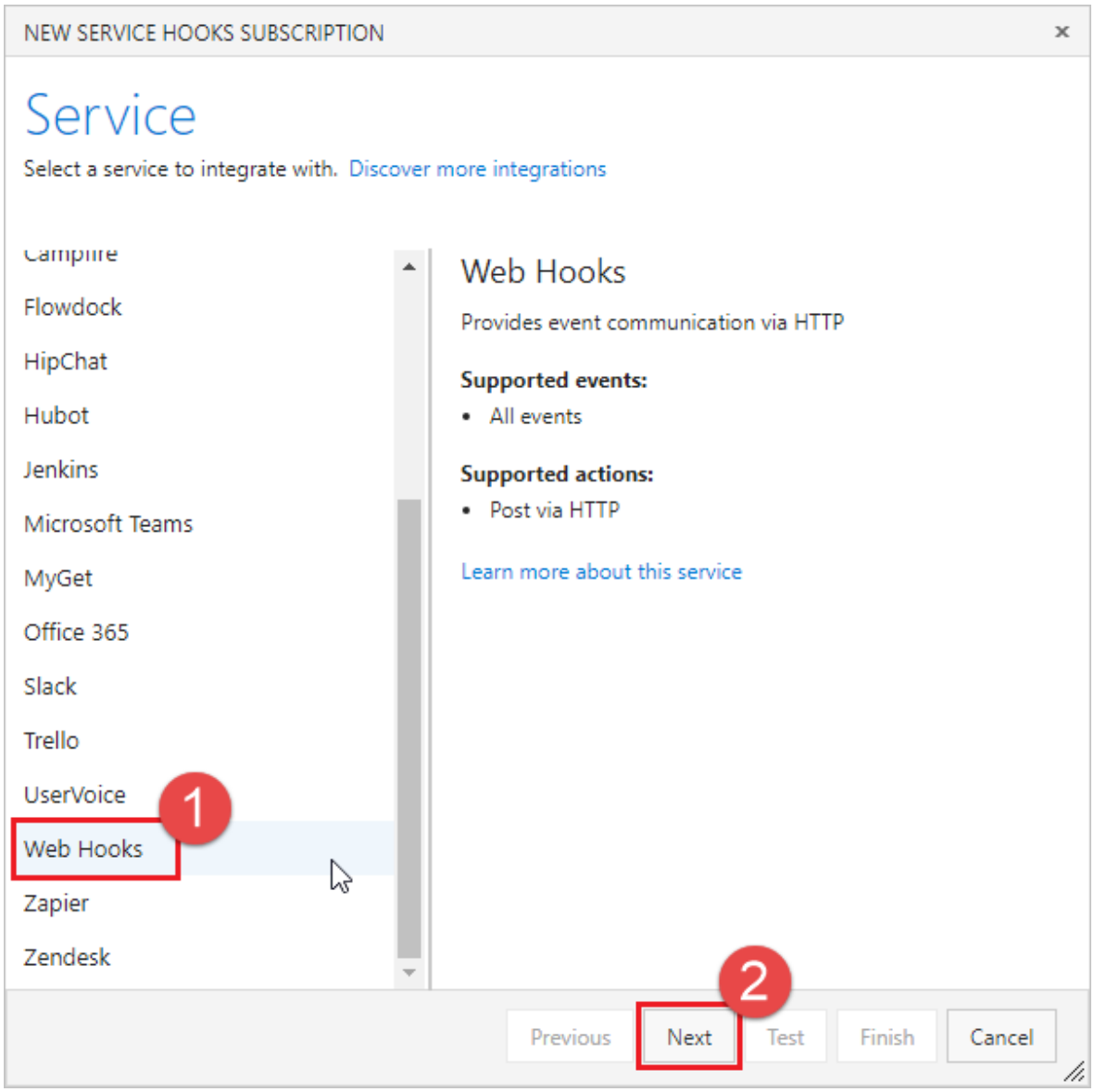
5. If this is your first service hook, select + **Create subscription**.



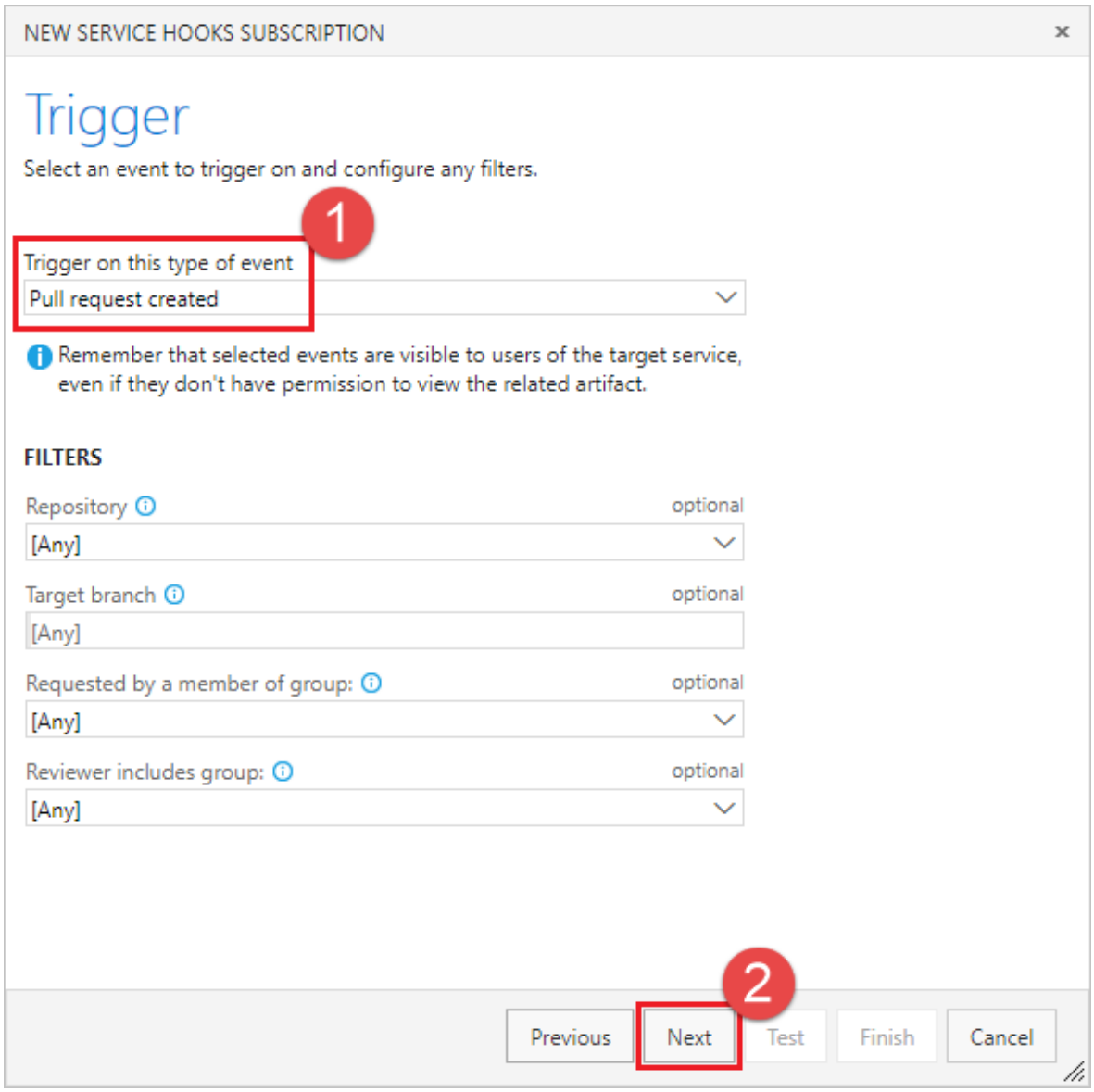
If you already have other service hooks configured, select the green plus `(+)` to create a new service hook subscription.



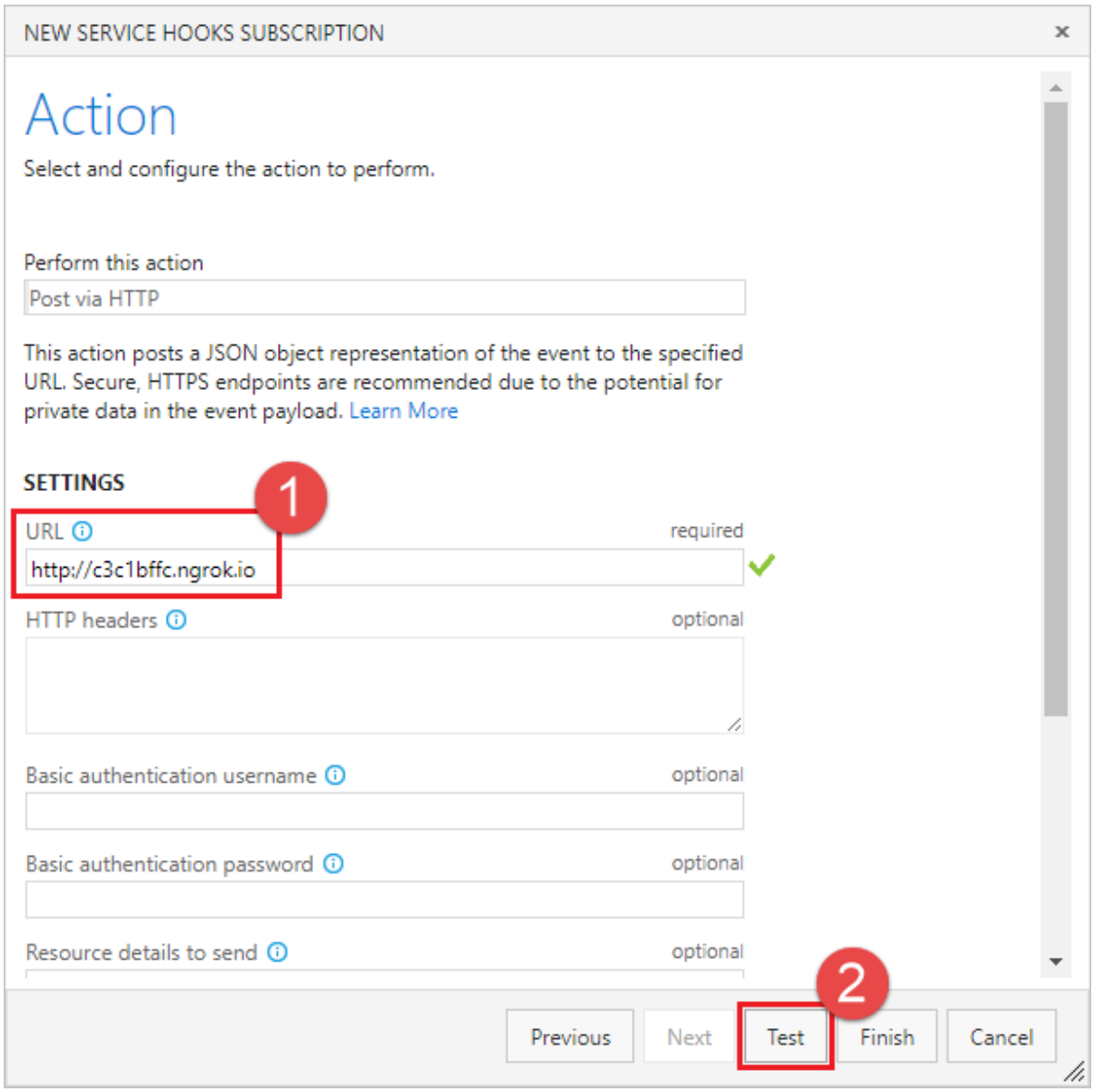
6. On the New Service Hooks Subscription dialog, select **Web Hooks** from the list of services, then select **Next**.



7. Select **Pull request created** from the list of event triggers, then select **Next**.



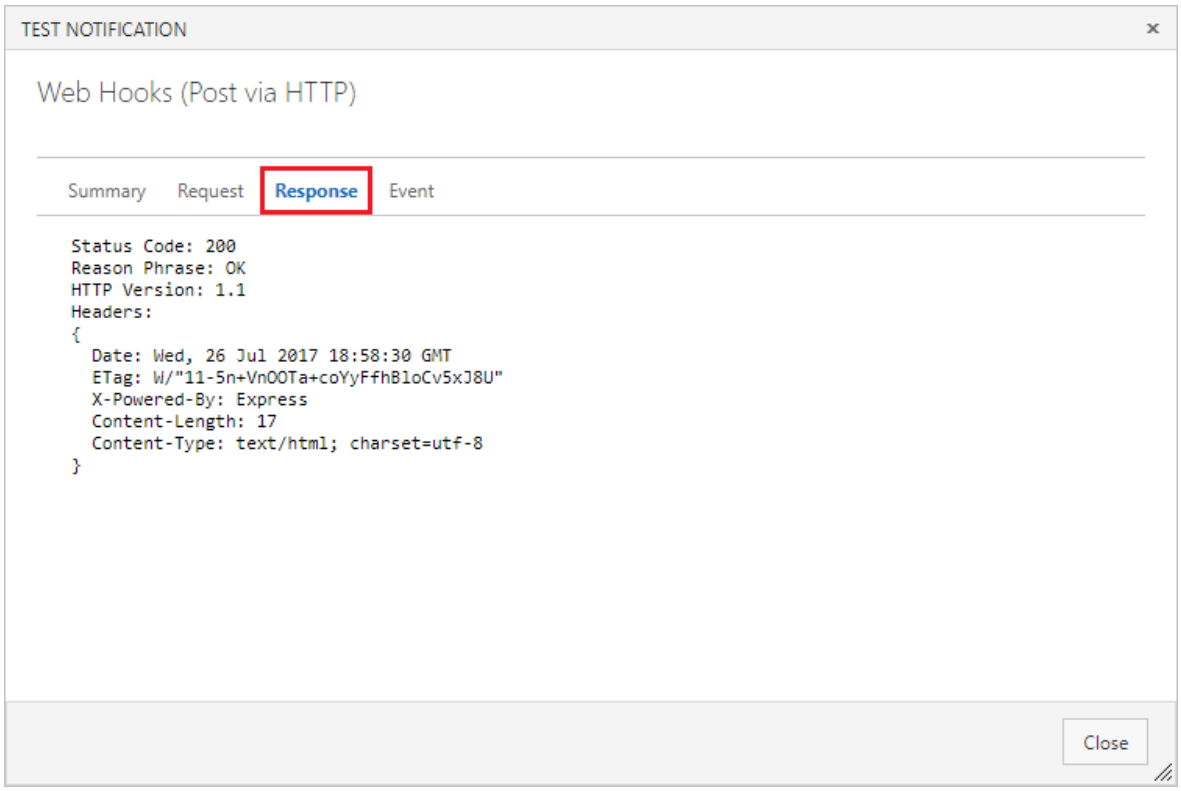
8. In the Action page, enter the URL from ngrok in the **URL** box. Select **Test** to send a test event to your server.



In the ngrok console window, you'll see an incoming `POST` that returned a `200 OK`, indicating your server received the service hook event.

```
HTTP Requests
-------------

POST /                          200 OK
```

In the Test Notification window, select the Response tab to see the details of the response from your server. You should see a content length of 17 that matches the length of the string from your POST handler (i.e. "Received the POST").



9. Close the Test Notification window, and select **Finish** to create the service hook.

Go through steps 3-9 again but this time configure the **Pull request updated** event.

> **ⓘ Important**
>
> Be sure to go through the preceding steps twice and create service hooks for both the **Pull request created** and **Pull request updated** events.

Post status to PRs

Now that your server can receive service hook events when new PRs are created, update it to post back status to the PR.

1. Service hook requests include a JSON payload describing the event. To help parse the JSON returned by the service hook, install the body-parser package.

```
npm install body-parser
```

2. Update `app.js` to use body-parser for parsing `application/json`.

```
JavaScript
```

```
var bodyParser = require('body-parser')

app.use(bodyParser.json())
```

3. To simplify making REST API calls to Azure Repos, install the azure-devops-node-api package.

```
npm install azure-devops-node-api
```

4. Update `app.js` to use the azure-devops-node-api package, set up the details for a connection to your account, and get an instance of the Git API.

JavaScript

```
const vsts = require("azure-devops-node-api")

const collectionURL = process.env.COLLECTIONURL
const token = process.env.TOKEN

var authHandler = vsts.getPersonalAccessTokenHandler(token)
var connection = new vsts.WebApi(collectionURL, authHandler)

var vstsGit = connection.getGitApi().then(
    vstsGit => {
        vstsGit.createPullRequestStatus(prStatus, repoId,
pullRequestId).then( result => {
            console.log(result);
        },
        error => {
            console.log(error);
        })
    },
    error => {
        console.log(error);
    }
);
```

5. Create an environment variable for your collection URL, replacing `<your account>` with the name of your Azure DevOps organization.

```
setx COLLECTIONURL "https://dev.azure.com/<your account>"
```

6. Create a personal auth token (PAT) for your app to use, following these instructions: Authenticating with personal access tokens. You should create a new PAT for every service that you use to access your account, naming it appropriately.

7. Create an environment variable for your PAT.

```
setx TOKEN "yourtokengoeshere"
```

8. Update the `post()` function to read the PR details from the service hook payload. You'll need these values in order to post back status.

JavaScript

```
var repoId = req.body.resource.repository.id
var pullRequestId = req.body.resource.pullRequestId
var title = req.body.resource.title
```

9. Build the status object to post on the PR.

   `State` is an enum of type GitStatusState. Use `succeeded` to indicate that the PR has passed the status check and is ready to merge.

   The `description` is a string value that will be displayed to the user in the Status section and activity feed in the PR details view.

   The `targetUrl` is a URL that will be used to create a link for the description text in the Status section and activity feed. This is the place where users can go to get more information about the status, for example, a build report or test run. If no URL is specified, the description will appear as text with no link.

   The context `name` and `genre` are used to categorize the status and distinguish it from other services posting status.

JavaScript

```
    var prStatus = {
        "state": "succeeded",
        "description": "Ready for review",
        "targetUrl": "https://visualstudio.microsoft.com",
        "context": {
            "name": "wip-checker",
            "genre": "continuous-integration"
        }
    }
```

10. Instead of just blindly posting the `succeeded` status, inspect the PR title to see if the user has indicated if the PR is a work in progress by adding **WIP** to the title. If so, change the status posted back to the PR.

```
JavaScript

    if (title.includes("WIP")) {
        prStatus.state = "pending"
        prStatus.description = "Work in progress"
    }
```

11. Finally, post the status using the `createPullRequestStatus()` method. It requires the status object, the repo ID, and the pull request ID. Output the response to the node console so you can see the result of the post.

```
JavaScript

vstsGit.createPullRequestStatus(prStatus, repoId, pullRequestId).then(
result => {
    console.log(result)
})
```

12. The resulting method should look something like this:

```
JavaScript

app.post("/", function (req, res) {

    // Get the details about the PR from the service hook payload
    var repoId = req.body.resource.repository.id
    var pullRequestId = req.body.resource.pullRequestId
    var title = req.body.resource.title

    // Build the status object that we want to post.
    // Assume that the PR is ready for review...
    var prStatus = {
        "state": "succeeded",
        "description": "Ready for review",
        "targetUrl": "https://visualstudio.microsoft.com",
        "context": {
            "name": "wip-checker",
            "genre": "continuous-integration"
        }
    }

    // Check the title to see if there is "WIP" in the title.
    if (title.includes("WIP")) {
```

```
        // If so, change the status to pending and change the
description.
        prStatus.state = "pending"
        prStatus.description = "Work in progress"
    }

    // Post the status to the PR
    vstsGit.createPullRequestStatus(prStatus, repoId,
pullRequestId).then( result => {
        console.log(result)
    })

    res.send("Received the POST")
})
```

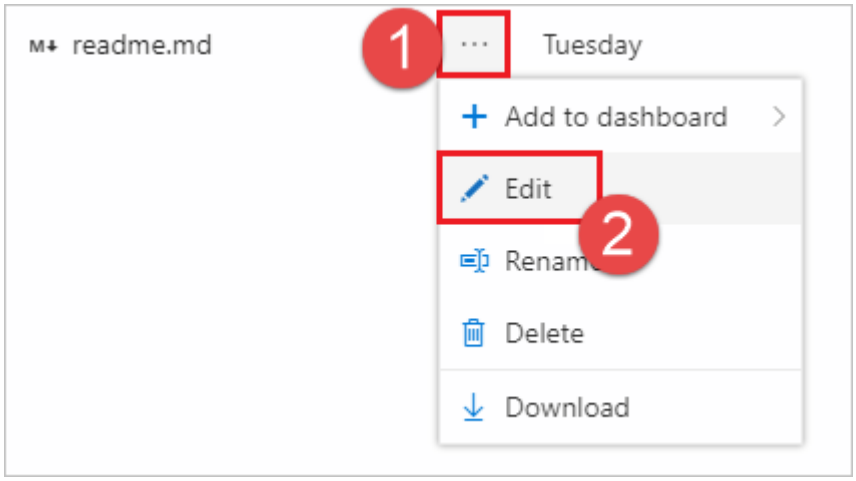
13. Save `app.js` and restart your node app.

```
node app.js
```

Create a new PR to test the status server

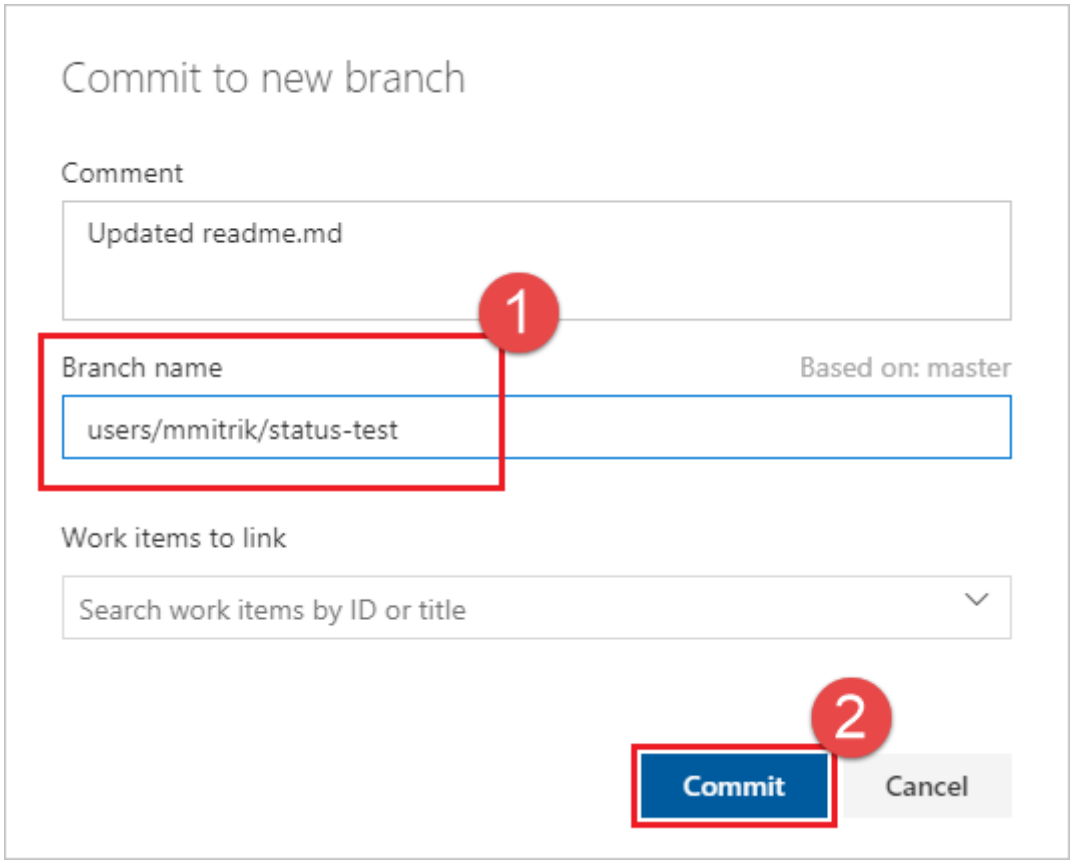
Now that your server is running and listening for service hook notifications, create a pull request to test it out.

1. Start in the files view. Edit the readme.md file in your repo (or any other file if you don't have a readme.md).

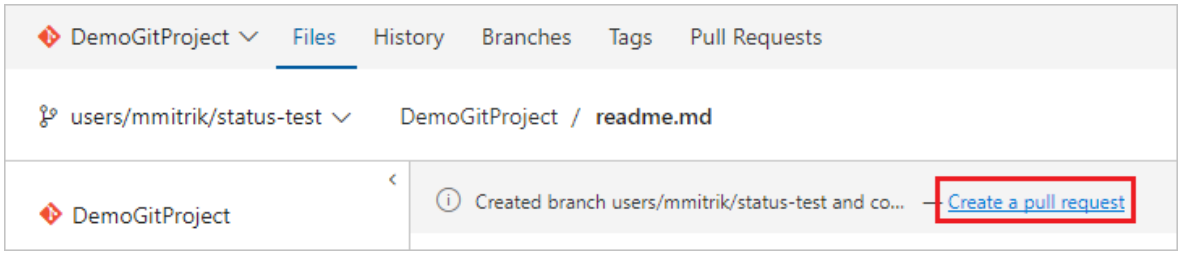


2. Make an edit and commit the changes to the repo.

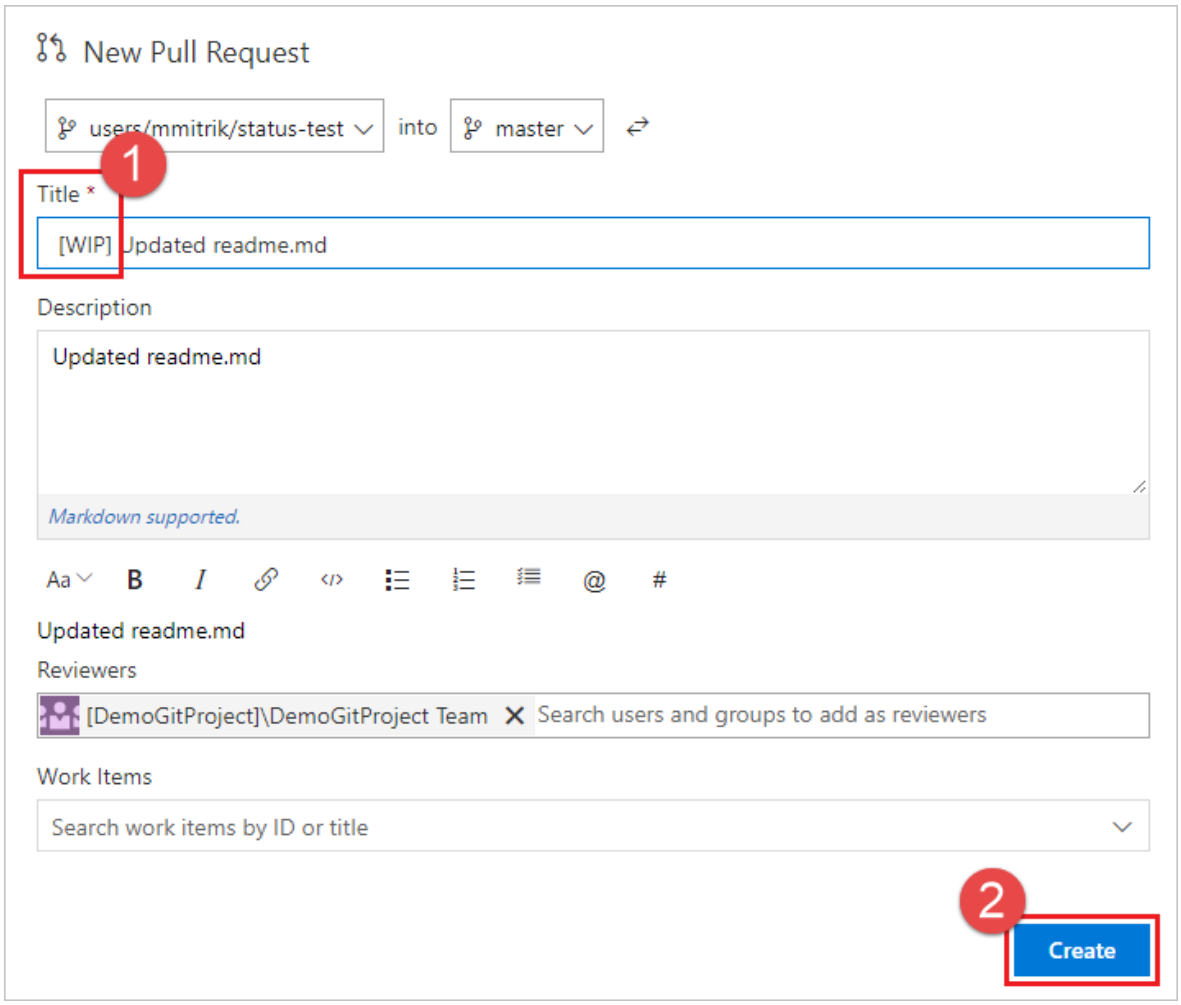
3. Be sure to commit the changes to a new branch so you can create a PR in the next step.



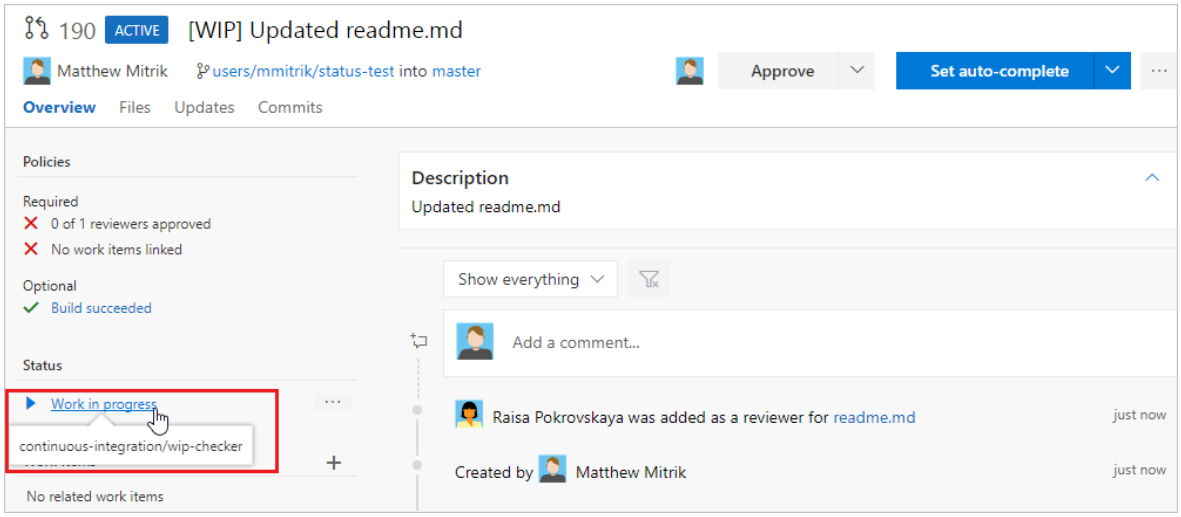
4. Select the **Create a pull request** link.



5. Add **WIP** in the title to test the functionality of the app. Select **Create** to create the PR.



6. Once the PR has been created, you will see the status section, with the **Work in progress** entry which links to the URL specified in the payload.



7. Update the PR title and remove the **WIP** text and note that the status changes from **Work in progress** to **Ready for review**.

Next Steps

- In this article, you learned the basics of how to create a service that listens for PR events via service hooks and can post status messages using the status API. For more information about the pull request status API see the REST API documentation.
- Configure a branch policy for an external service.

Use Azure Functions to create custom branch policies

Article • 10/04/2022

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019 | TFS 2018

The pull request (PR) workflow provides developers with an opportunity to get feedback on their code from peers as well as from automated tools. 3rd party tools and services can participate in the PR workflow by using the PR Status API. This article guides you through the process of creating a custom branch policy using Azure Functions to validate PRs in an Azure DevOps Services Git repository. With Azure Functions you don't have to worry about provisioning and maintaining servers, especially when your workload grows. Azure Functions provide a fully managed compute platform with high reliability and security.

For more information about PR status, see Customize and extend pull request workflows with pull request status.

Prerequisites

An organization in Azure DevOps with a Git repo. If you don't have an organization, sign up to upload and share code in free unlimited private Git repositories.

Create a basic Azure function to listen to Azure Repos events

Follow the create your first Azure function documentation to create a simple function. Modify the code in the sample to look like this:

C#

```
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using Newtonsoft.Json;

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req,
TraceWriter log)
```

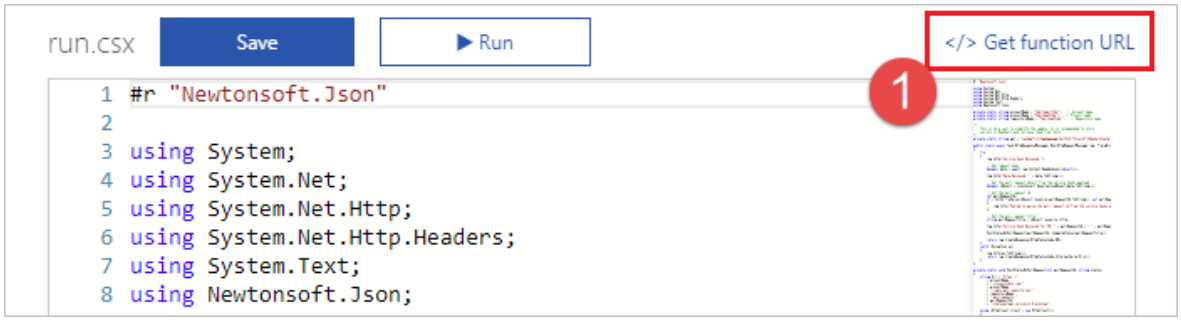
```
{
    try
    {
        log.Info("Service Hook Received.");

        // Get request body
        dynamic data = await req.Content.ReadAsAsync<object>();

        log.Info("Data Received: " + data.ToString());

        // Get the pull request object from the service hooks payload
        dynamic jObject = JsonConvert.DeserializeObject(data.ToString());

        // Get the pull request id
        int pullRequestId;
        if (!Int32.TryParse(jObject.resource.pullRequestId.ToString(), out
pullRequestId))
        {
            log.Info("Failed to parse the pull request id from the service
hooks payload.");
        };

        // Get the pull request title
        string pullRequestTitle = jObject.resource.title;

        log.Info("Service Hook Received for PR: " + pullRequestId + " " +
pullRequestTitle);

        return req.CreateResponse(HttpStatusCode.OK);
    }
    catch (Exception ex)
    {
        log.Info(ex.ToString());
        return req.CreateResponse(HttpStatusCode.InternalServerError);
    }
}
```

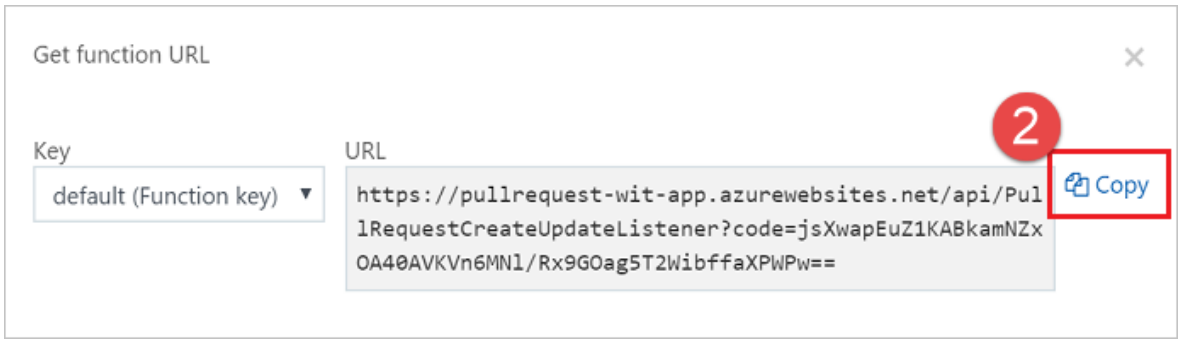
Configure a service hook for PR events

Service hooks are an Azure DevOps Services feature that can alert external services when certain events occur. For this sample, you'll want to set up a service hook for PR events, your Azure function will be notified when a pull request changes. In order to receive `POST` requests when pull requests change, you will need to provide the service hook with the Azure function URL.

For this sample you will need to configure 2 service hooks. The first will be for the **Pull request created** event and the second will be for the **Pull request updated** event.

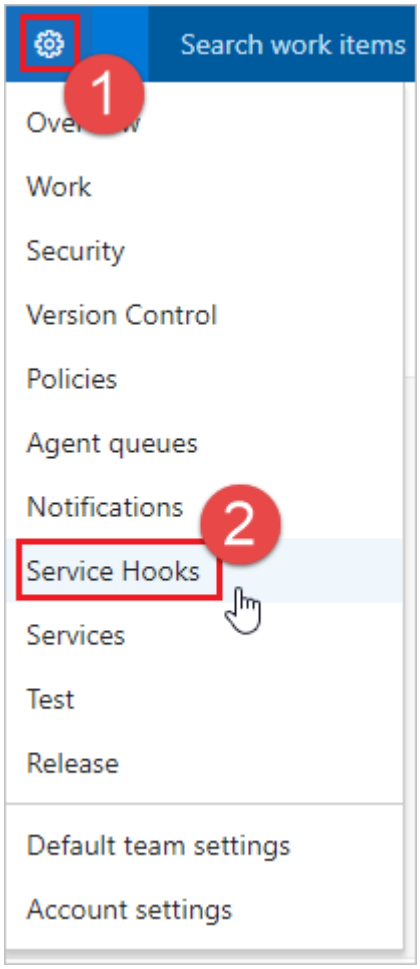
1. Get the function URL from the Azure portal by clicking the **Get function URL** in your Azure function view and copy the URL.



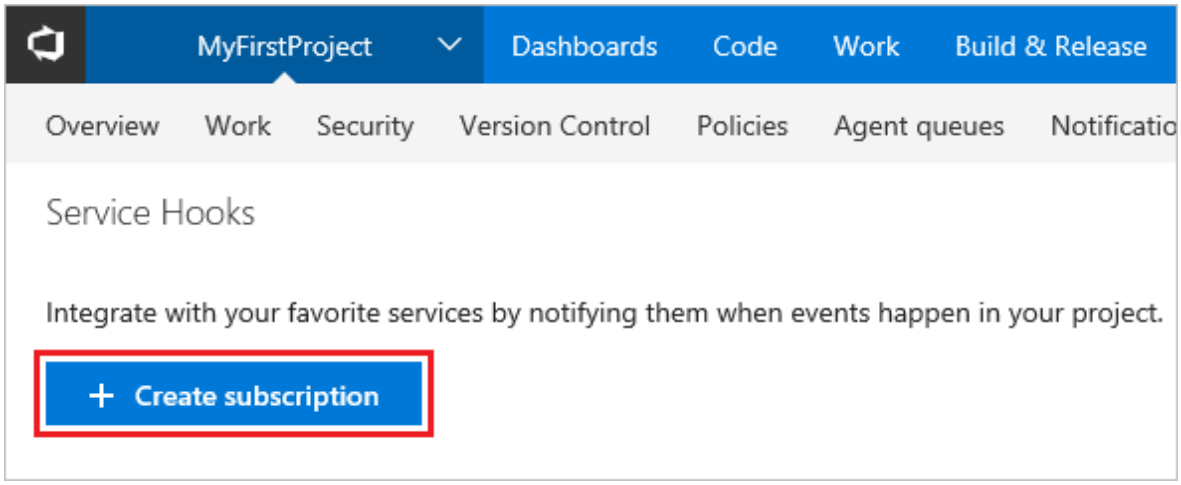


2. Browse to your project in Azure DevOps, e.g. `https://dev.azure.com/<your organization>/<your project name>`

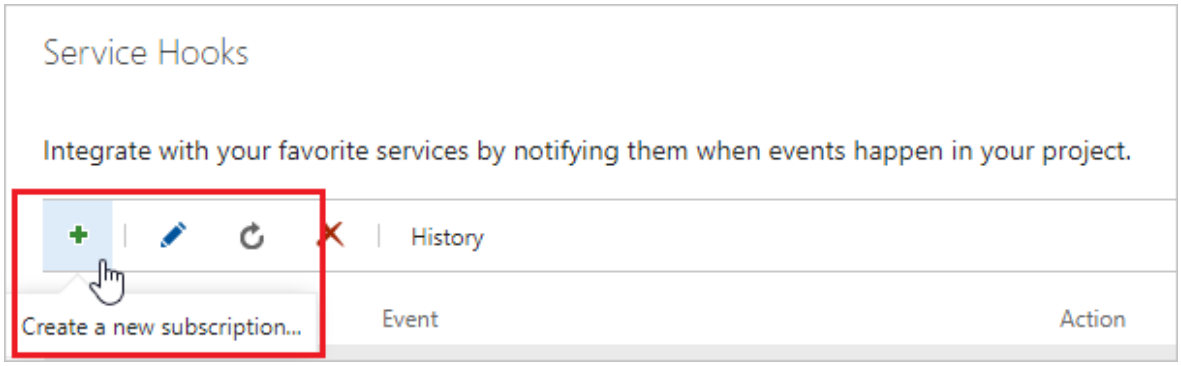
3. From the navigation menu, hover over the **gear** and select **Service Hooks**.



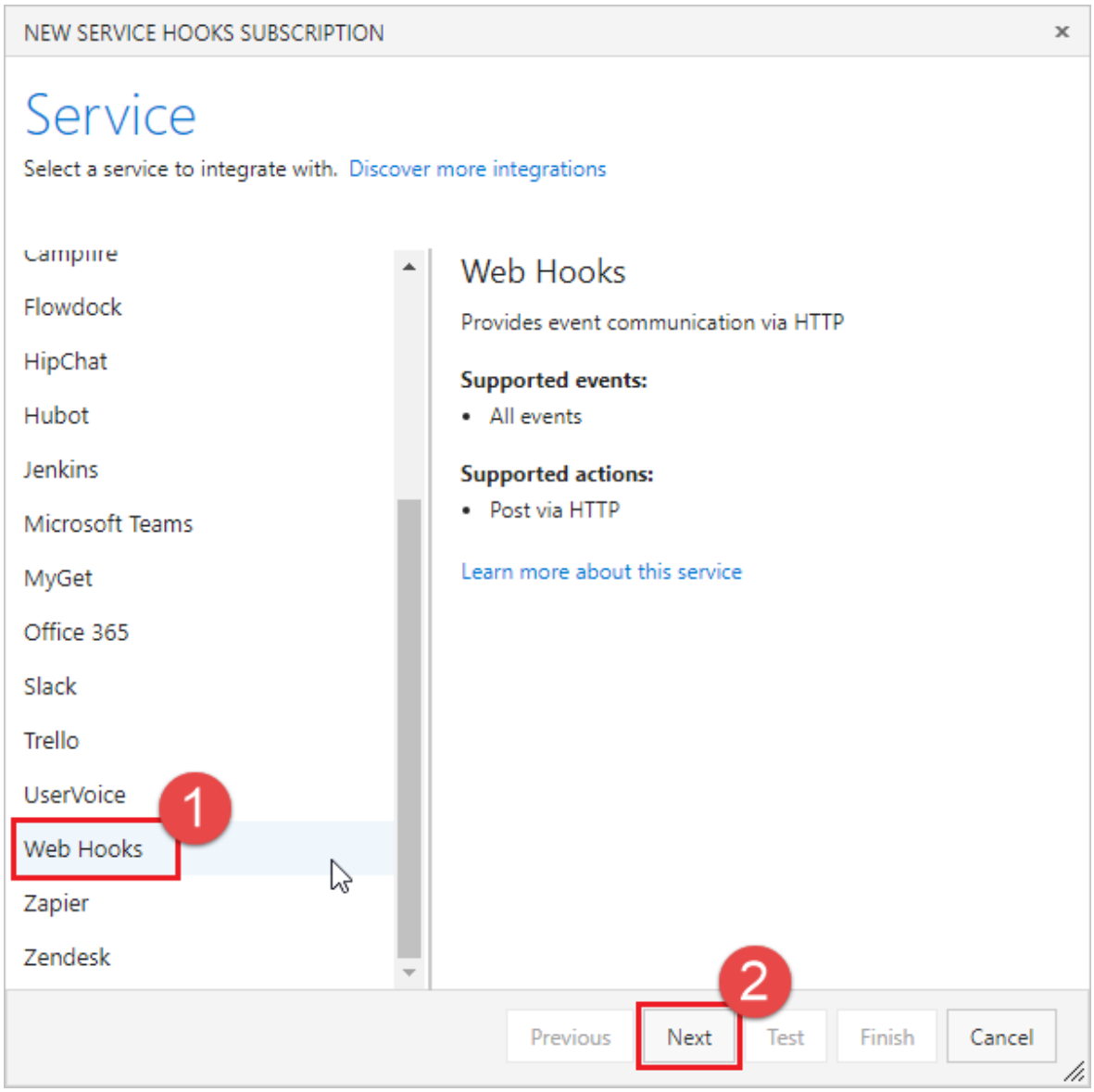
4. If this is your first service hook, select + **Create subscription**.



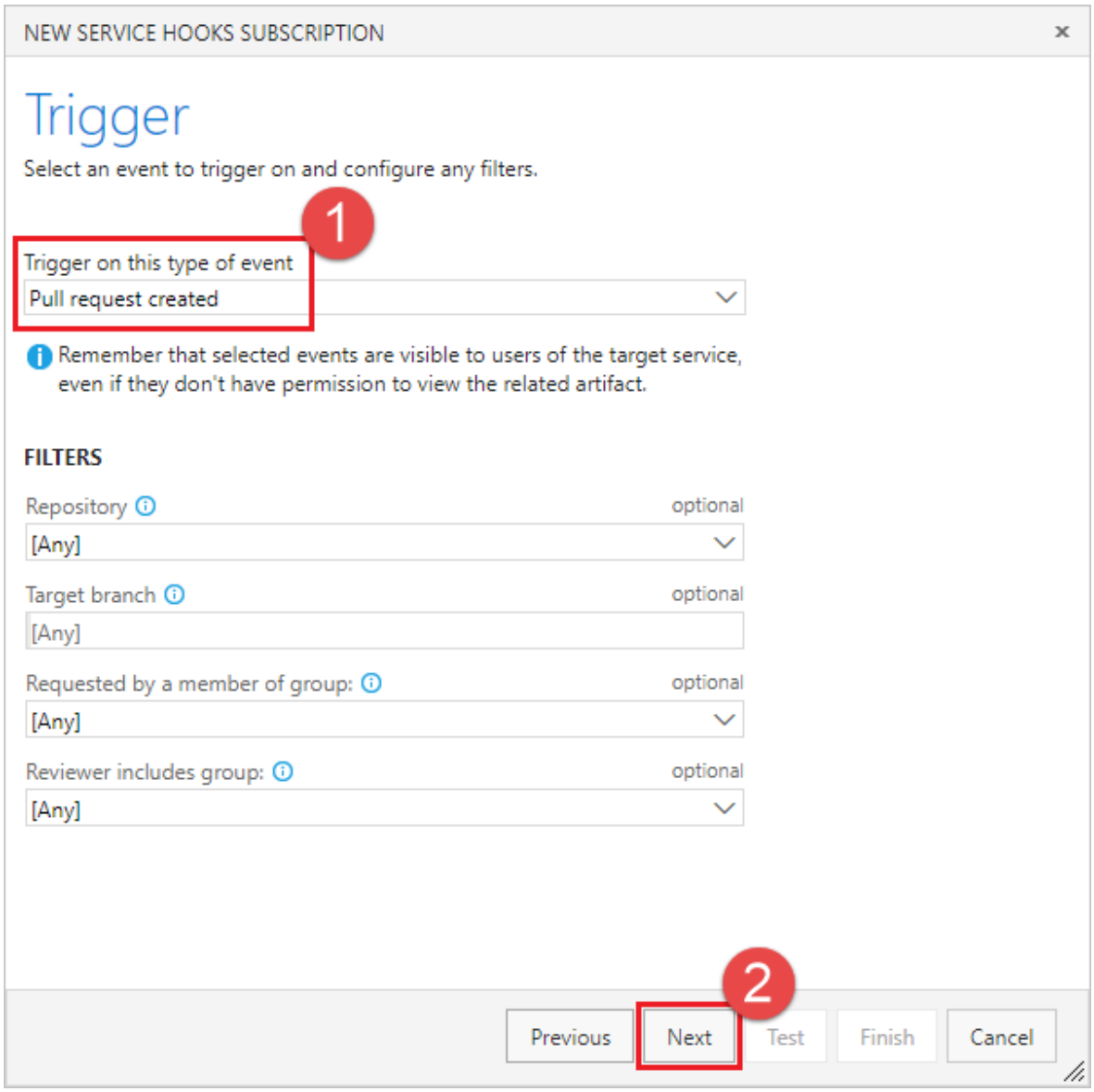
If you already have other service hooks configured, select the green plus `(+)` to create a new service hook subscription.



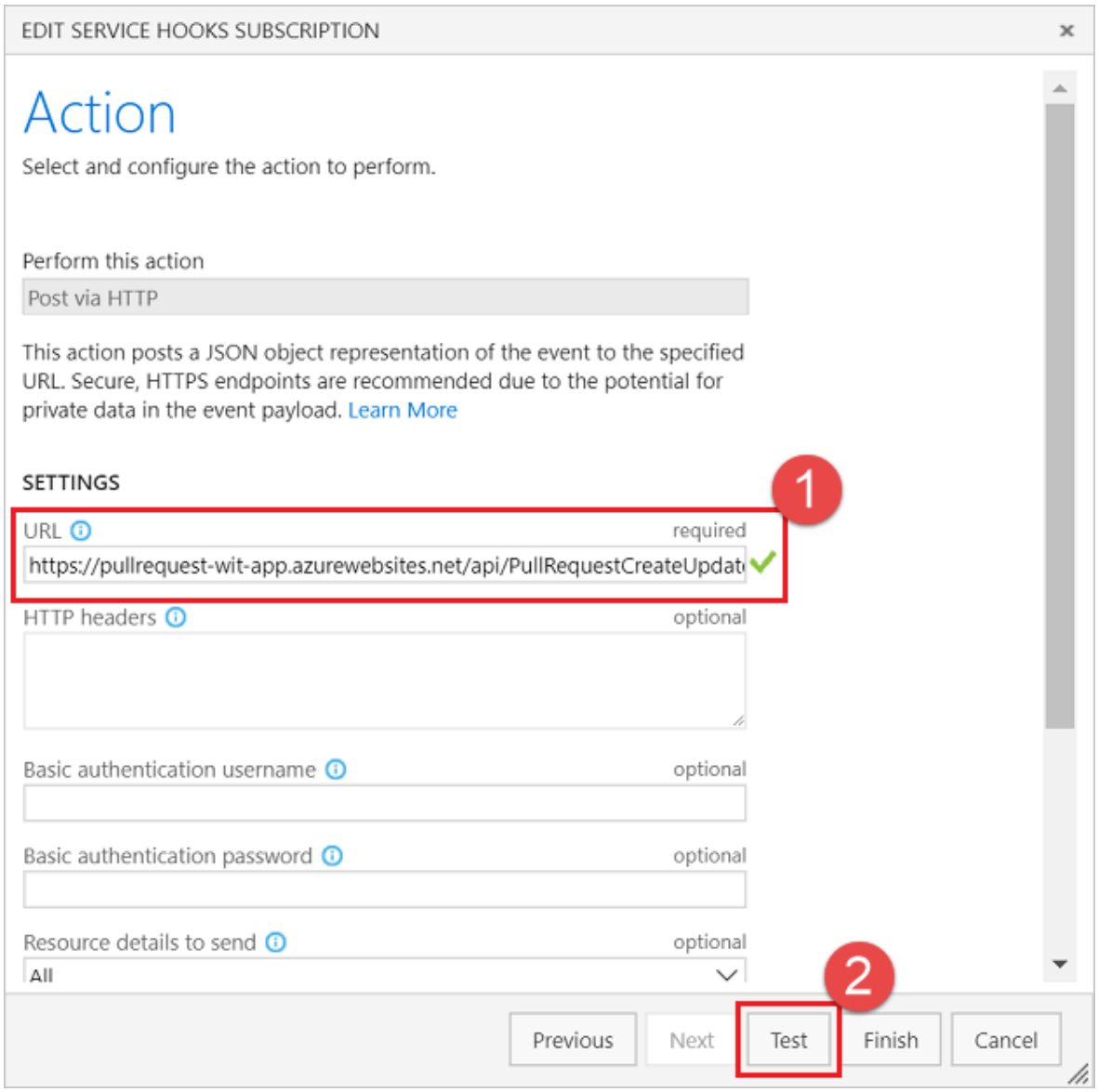
5. On the New Service Hooks Subscription dialog, select **Web Hooks** from the list of services, then select **Next**.



6. Select **Pull request created** from the list of event triggers, then select **Next**.



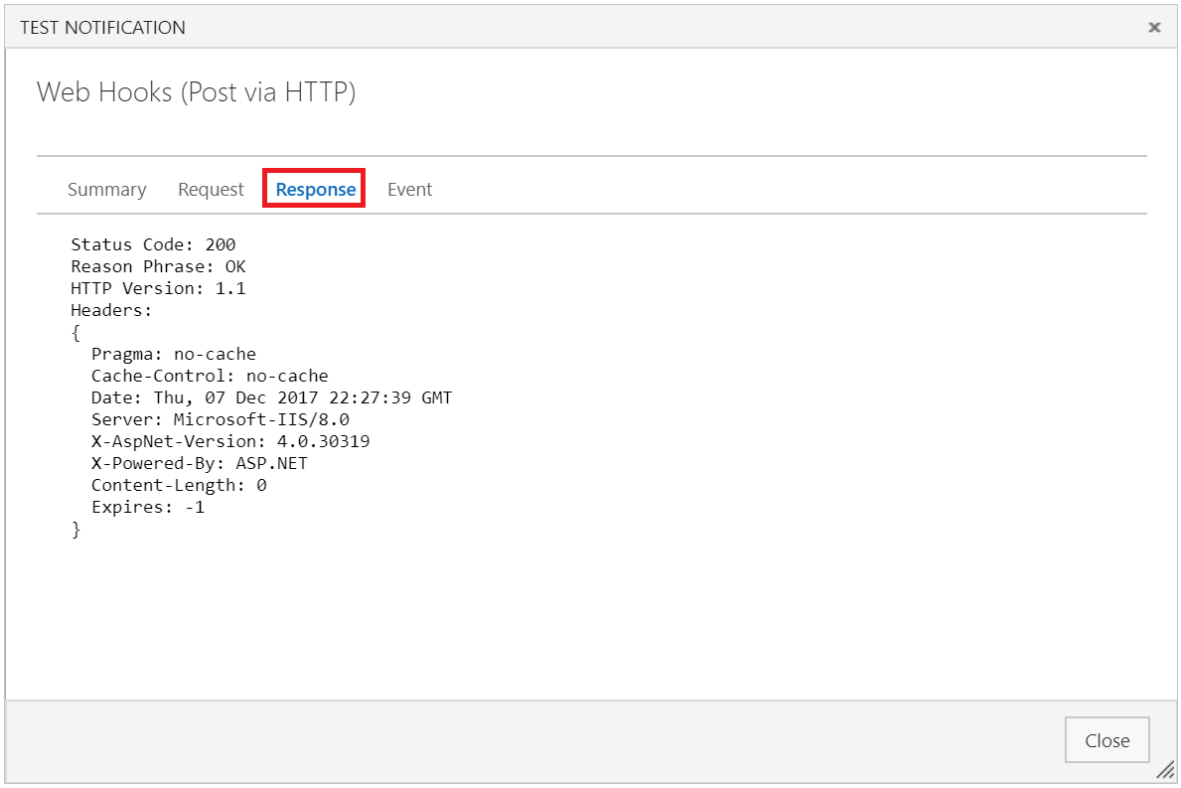
7. In the Action page, enter the URL that you copied in step 1 in the **URL** box. Select **Test** to send a test event to your server.



In the Azure function log window, you'll see an incoming `POST` that returned a `200 OK`, indicating your function received the service hook event.

```
HTTP Requests
-------------

POST /                          200 OK
```

In the Test Notification window, select the Response tab to see the details of the response from your server. You should see the response from your server.



8. Close the Test Notification window, and select **Finish** to create the service hook.

Go through steps 2-8 again but this time configure the **Pull request updated** event.

> **ⓘ Important**
>
> Be sure to go through the preceding steps twice and create service hooks for both the **Pull request created** and **Pull request updated** events.

Create a pull request to verify your Azure function is receiving notifications.

Post status to PRs

Now that your server can receive service hook events when new PRs are created, update it to post back status to the PR. You can use the JSON payload posted by the service hook in order to determine what status to set on your PR.

Update the code of your Azure function to look like the following example.

Make sure to update the code with your organization name, project name, repository name and PAT token. In order to have permission to change PR status, the PAT requires vso.code_status scope, which you can grant by selecting the **Code (status)** scope on the **Create a personal access token** page.

> **ⓘ Important**

This sample code stores the PAT in code to simplify the sample. It is recommended to store secrets in KeyVault and retrieve them from there.

This sample inspects the PR title to see if the user has indicated if the PR is a work in progress by adding **WIP** to the title. If so, the sample code changes the status posted back to the PR. Replace the code in your Azure function with the following code to implement updating the status posted back to the PR.

C#

```
using System;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;
using Newtonsoft.Json;

private static string organizationName = "[Organization Name]";  //
Organization name
private static string projectName      = "[Project Name]";       // Project
name
private static string repositoryName   = "[Repo Name]";          //
Repository name

/*
    This is here just to simplify the sample, it is recommended to store
    secrets in KeyVault and retrieve them from there.
*/
private static string pat = "[PAT TOKEN]";

public static async Task<HttpResponseMessage> Run(HttpRequestMessage req,
TraceWriter log)
{
    try
    {
        log.Info("Service Hook Received.");

        // Get request body
        dynamic data = await req.Content.ReadAsAsync<object>();

        log.Info("Data Received: " + data.ToString());

        // Get the pull request object from the service hooks payload
        dynamic jObject = JsonConvert.DeserializeObject(data.ToString());

        // Get the pull request id
        int pullRequestId;
        if (!Int32.TryParse(jObject.resource.pullRequestId.ToString(), out
pullRequestId))
        {
            log.Info("Failed to parse the pull request id from the service
hooks payload.");
```

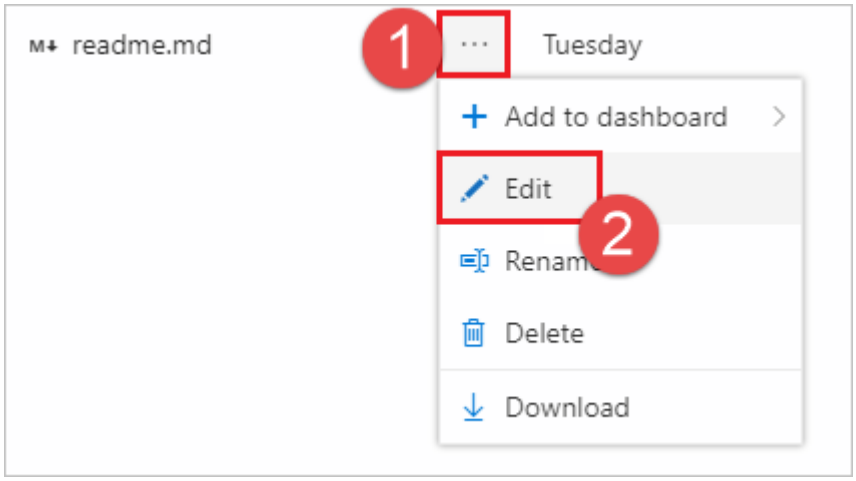
```
        };

        // Get the pull request title
        string pullRequestTitle = jObject.resource.title;

        log.Info("Service Hook Received for PR: " + pullRequestId + " " +
pullRequestTitle);

        PostStatusOnPullRequest(pullRequestId,
ComputeStatus(pullRequestTitle));

        return req.CreateResponse(HttpStatusCode.OK);
    }
    catch (Exception ex)
    {
        log.Info(ex.ToString());
        return req.CreateResponse(HttpStatusCode.InternalServerError);
    }
}

private static void PostStatusOnPullRequest(int pullRequestId, string
status)
{
    string Url = string.Format(

@"https://dev.azure.com/{0}/{1}/_apis/git/repositories/{2}/pullrequests/{3}/
statuses?api-version=4.1",
        organizationName,
        projectName,
        repositoryName,
        pullRequestId);

    using (HttpClient client = new HttpClient())
    {
        client.DefaultRequestHeaders.Accept.Add(new
MediaTypeWithQualityHeaderValue("application/json"));
        client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic", Convert.ToBase64String(
                ASCIIEncoding.ASCII.GetBytes(
                string.Format("{0}:{1}", "", pat))));

        var method = new HttpMethod("POST");
        var request = new HttpRequestMessage(method, Url)
        {
            Content = new StringContent(status, Encoding.UTF8,
"application/json")
        };

        using (HttpResponseMessage response =
client.SendAsync(request).Result)
        {
            response.EnsureSuccessStatusCode();
        }
    }
}
```

```
private static string ComputeStatus(string pullRequestTitle)
{
    string state = "succeeded";
    string description = "Ready for review";

    if (pullRequestTitle.ToLower().Contains("wip"))
    {
        state = "pending";
        description = "Work in progress";
    }

    return JsonConvert.SerializeObject(
        new
        {
            State = state,
            Description = description,
            TargetUrl = "https://visualstudio.microsoft.com",

            Context = new
            {
                Name = "PullRequest-WIT-App",
                Genre = "pr-azure-function-ci"
            }
        });
}
```

Create a new PR to test the status server

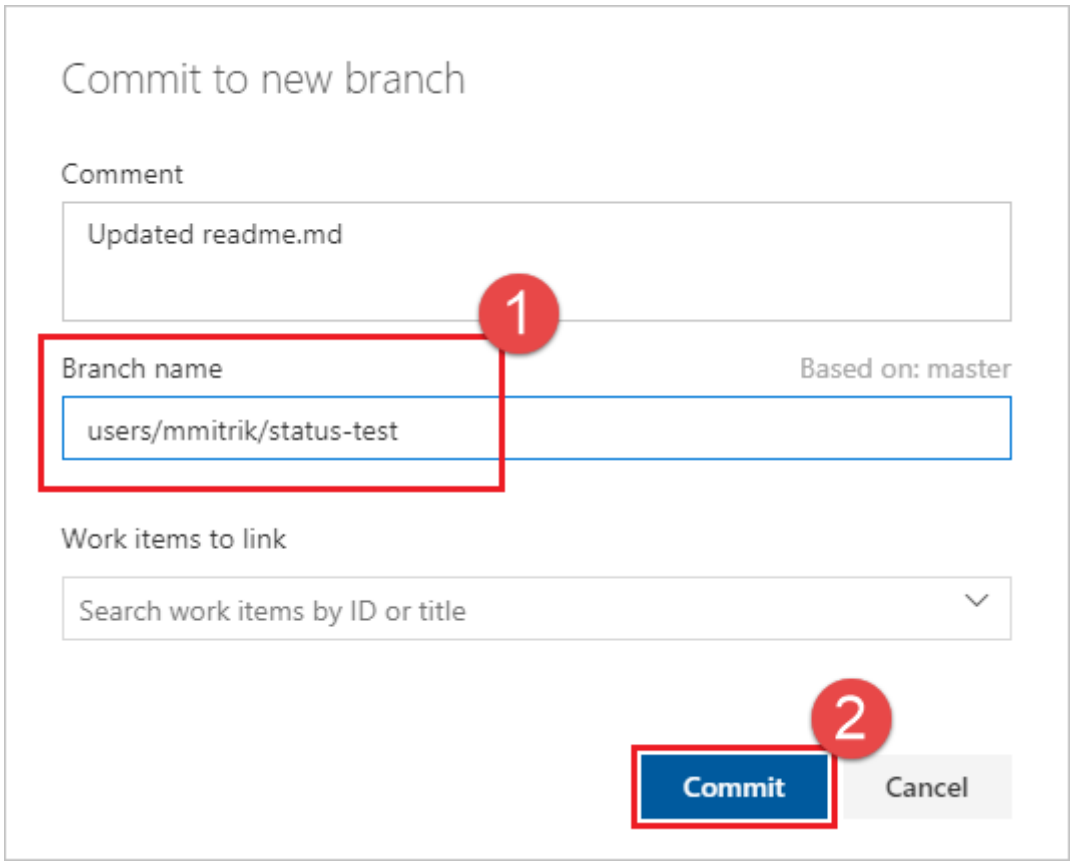
Now that your server is running and listening for service hook notifications, create a pull request to test it out.

1. Start in the files view. Edit the readme.md file in your repo (or any other file if you don't have a readme.md).

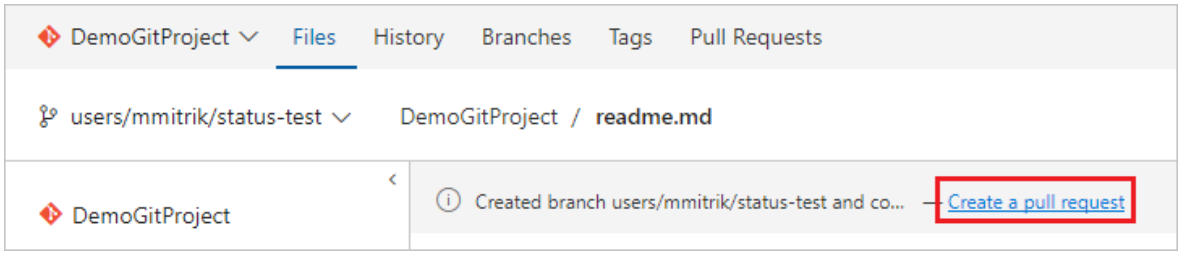


2. Make an edit and commit the changes to the repo.

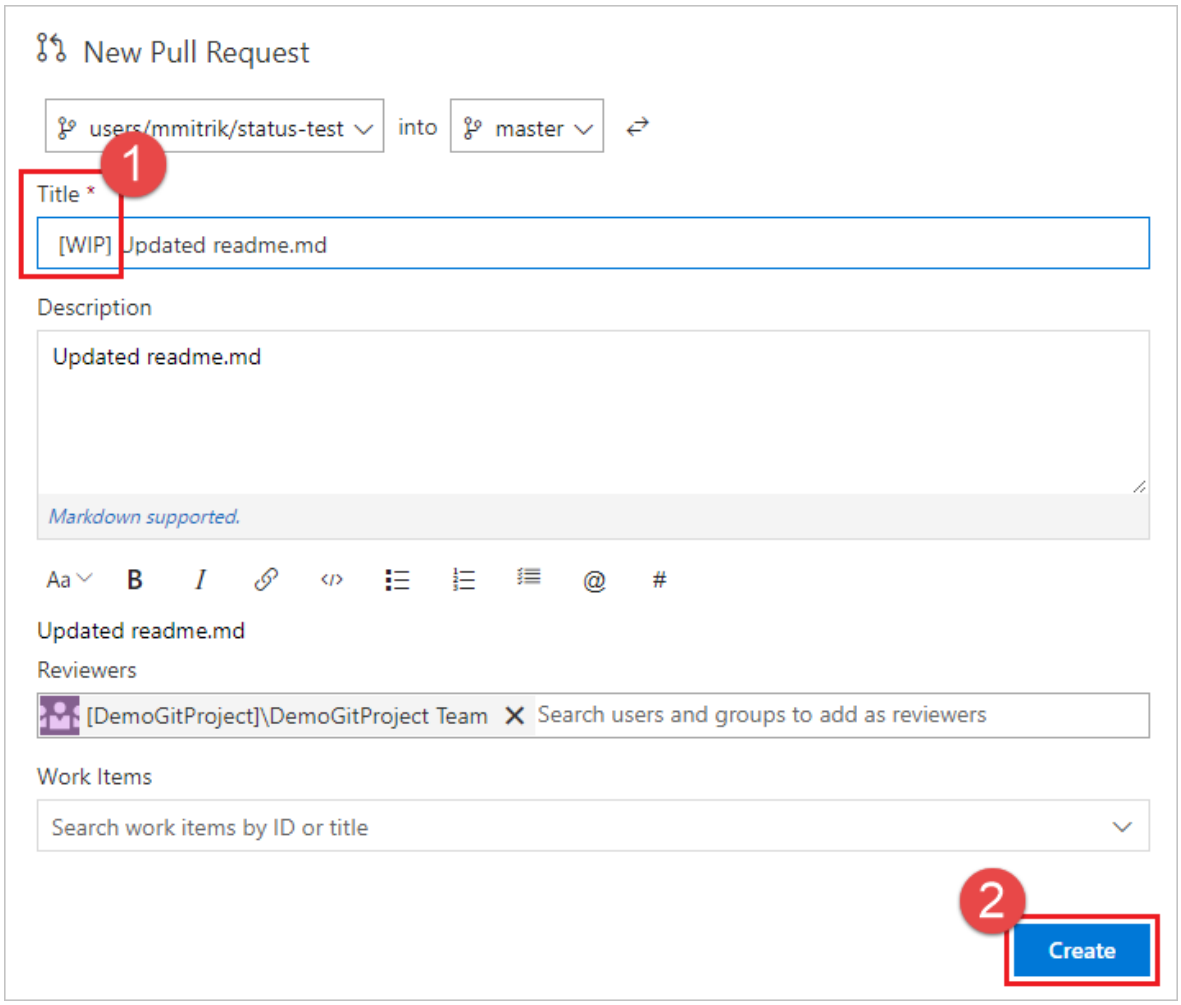
3. Be sure to commit the changes to a new branch so you can create a PR in the next step.



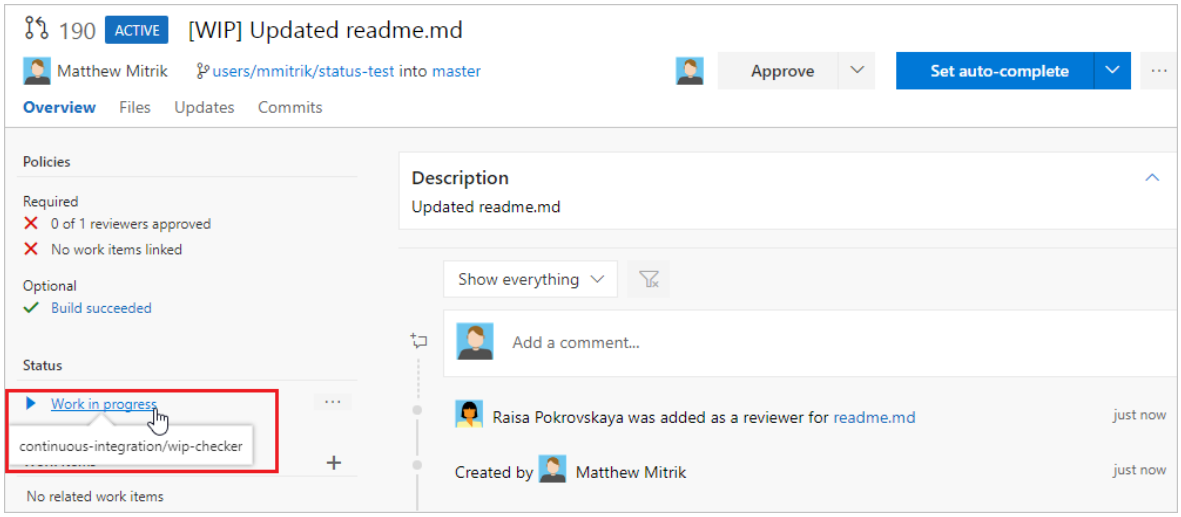
4. Select the **Create a pull request** link.



5. Add **WIP** in the title to test the functionality of the app. Select **Create** to create the PR.



6. Once the PR has been created, you will see the status section, with the **Work in progress** entry which links to the URL specified in the payload.



7. Update the PR title and remove the **WIP** text and note that the status changes from **Work in progress** to **Ready for review**.

Next Steps

- In this article, you learned the basics of how to create a serverless Azure function that listens for PR events via service hooks and can post status messages using the status API. For more information about the pull request status API see the REST API documentation.
- Configure a branch policy for an external service.

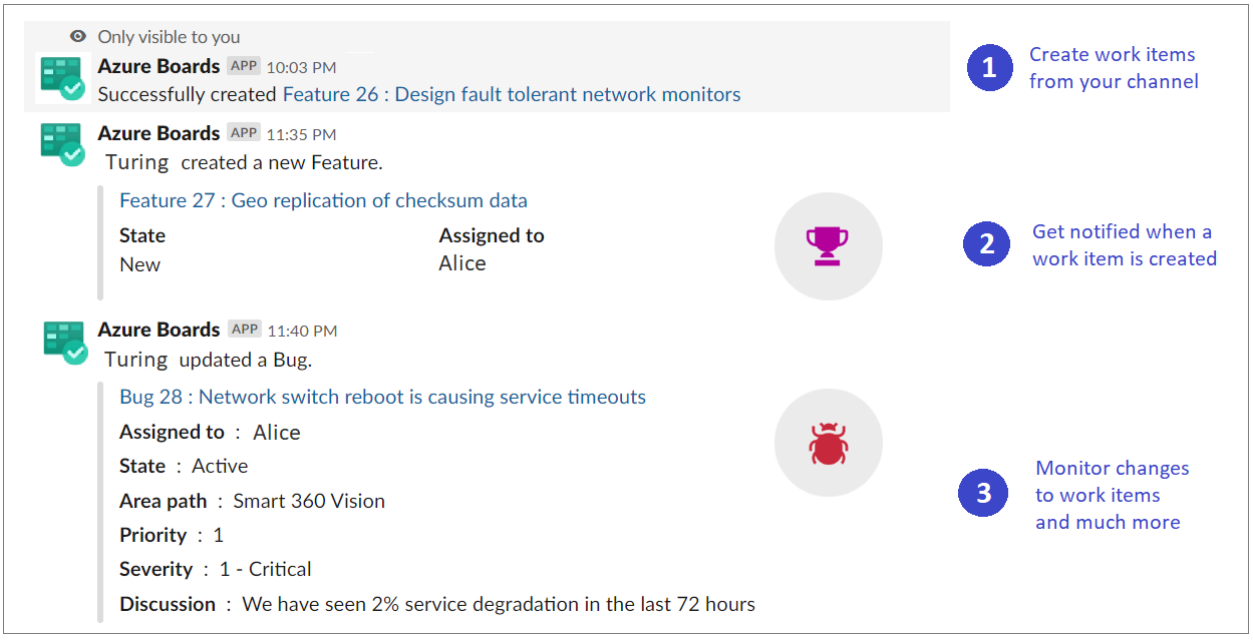
Use the Azure Boards app with Slack to manage work items

Article • 11/21/2024

Azure DevOps Services

If you use Slack, you can use the Azure Boards app for Slack to create work items and monitor work item activity in your Azure Boards project from your Slack channel.

The Azure Boards app for Slack allows users to set up and manage subscriptions in their Slack channel. They can manage subscriptions for create, update, and other work item events. Users can also get notifications for these events in their Slack channel. Conversations in the Slack channel can be used to create work items. Previews for work item URLs help users to start discussions around work.



> **Note**
>
> - Azure Boards and Slack integration is only supported for Azure DevOps Services.
> - Notifications are currently not supported inside direct messages.

Prerequisites

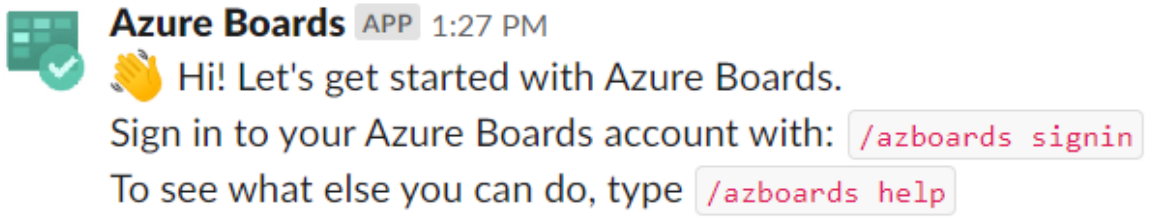
- **Access levels**: To create a work item, be a **Contributor** for the project.

- **Permissions**: To create subscriptions in a Slack channel for work item events, be a member of the Project Administrators group or Team Administrators group.
- **Policies**: To receive notifications, enable the **Third party application access via OAuth** policy for the organization.

Add the Azure Boards app to your Slack workspace

1. To install the Azure Boards app to your Slack workspace, open a web browser, sign into Slack, and open the Azure Boards app.

2. Once added, a welcome message displays from the app as shown in the following image.

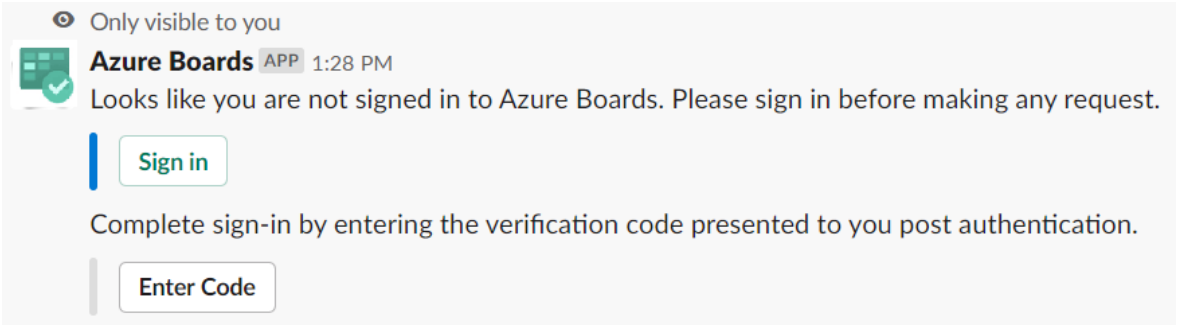


3. Use the `/azboards` Slack handle to interact with the app. A list of commands is provided later in this article, Command reference.

Link your Azure Boards project to the Azure Boards app

To use the app, you must first link your Azure Boards project to your Slack channel.

1. Once the app gets installed in your Slack workspace, connect and authenticate yourself to Azure Boards.



2. After signing in, use the following slash command inside a Slack channel to link to the Azure Boards project that you specify with the URL:

```
/azboards link [project url]
```

For example:

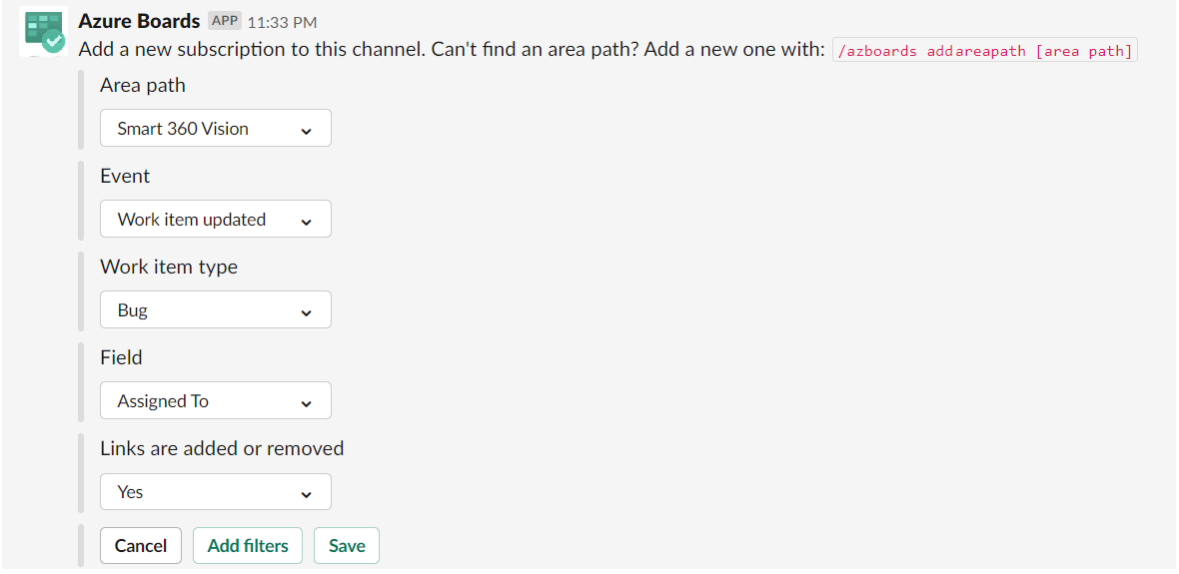
```
/azboards link https://dev.azure.com/myorg/myproject
```

Once the project is linked, you can create work items using `/azboards create` command or use message actions.

Set up subscriptions to monitor work items

You can create subscriptions to monitor work items at any time using the `/azboards subscriptions` command. You have an option of setting up subscriptions just after linking a project.

1. Select area path you want, the event that you're interested in, and use the associated filters to customize your Slack channel. To easily set up subscriptions, your recently accessed area paths are shown in the area path dropdown.



In case your team's area path doesn't appear in the Area path dropdown menu, follow the instructions mentioned in the next section, Add area paths. Area paths added using the `/azboards addAreapath` command and area paths for which subscriptions are created in the Slack channel always appear in the Area path dropdown along with recently accessed area paths.

Add area paths

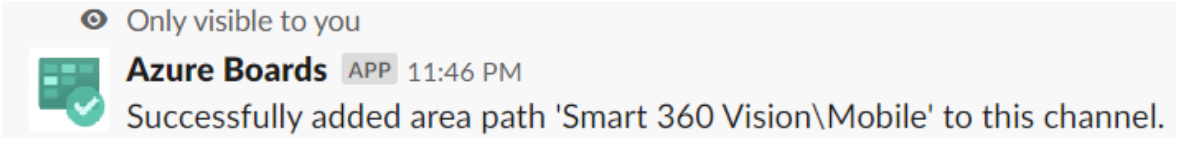
You can add areas that your team works on to the channel so that they're always available for creating work items and subscriptions. This action is important mainly for the teams with more than 100 area paths.

- Use the following command to add area paths from your project to the Slack channel.

  ```
  /azboards addAreapath [area path]
  ```

  For example:

  ```
  /azboards addAreapath myproject\fabrikam
  ```

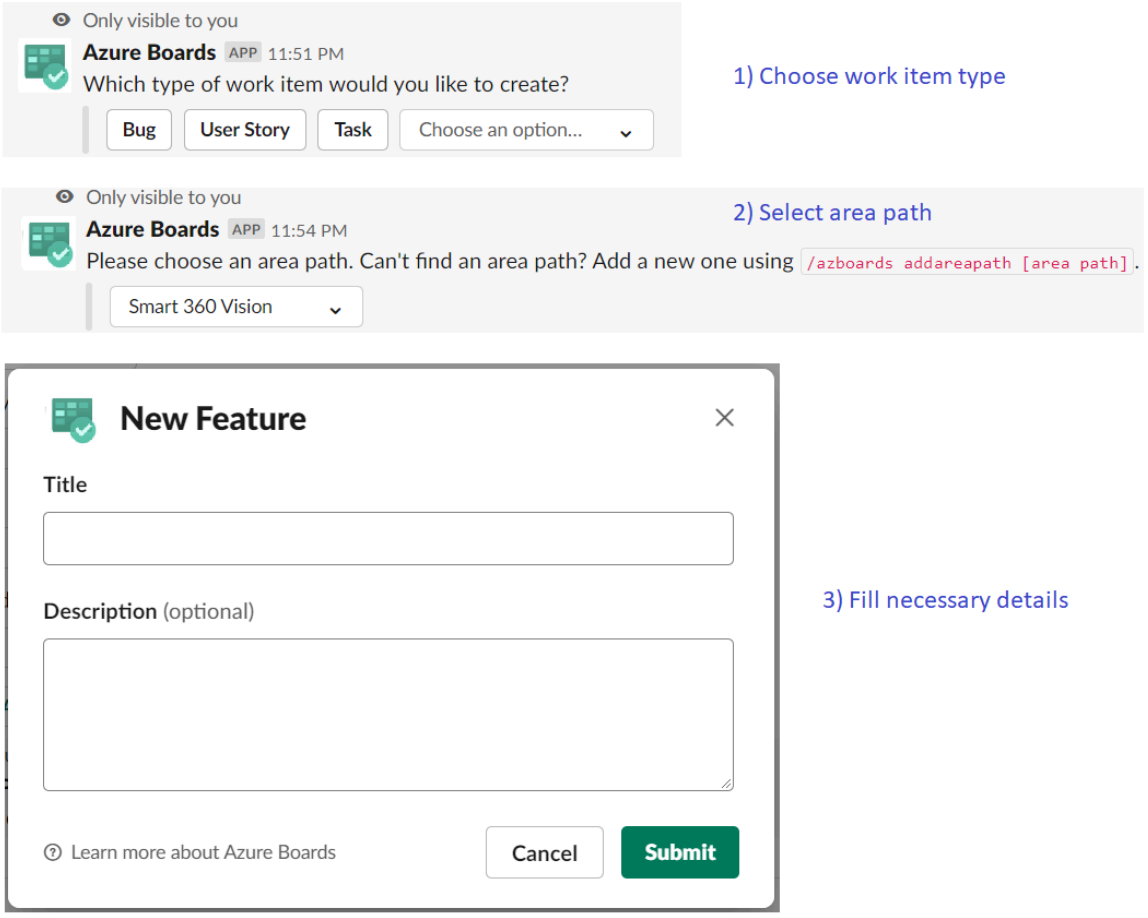
  

- If you choose project name as your area path, then you receive notifications for all the area paths in the project. It's logically equivalent to choosing 'Any' area path.

Create a work item with a command

1. With the Azure Boards app, you can create work items from your channel. The app supports custom work items as well.

- To create a work item, use `/azboards create`.



2. You can create work items directly from a command by passing work item type and title as parameters. Work items get created only if they don't have any fields to be mandatorily filled.

```
/azboards create [work item type] [work item title]
```

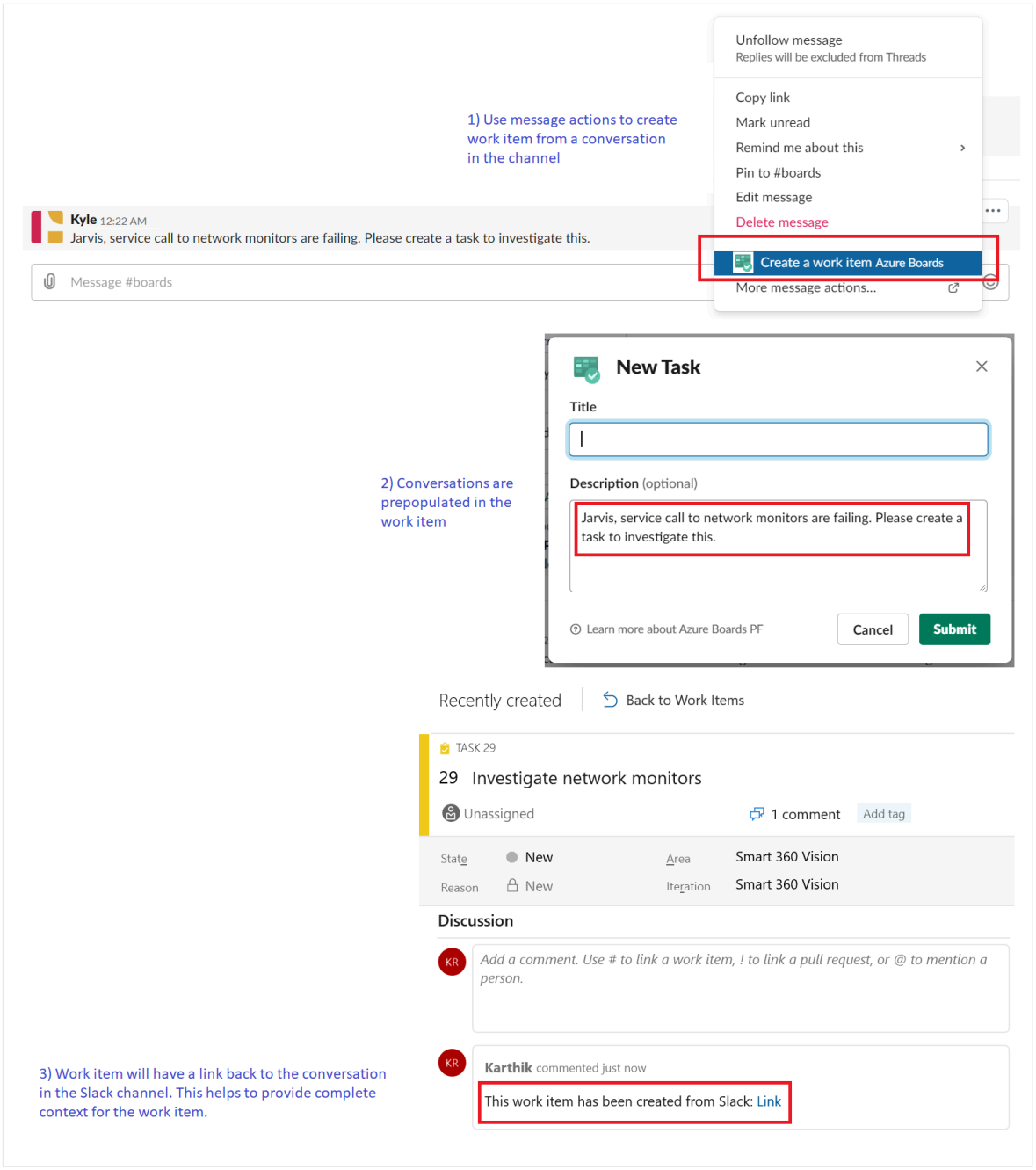
For example:

```
/azboards create 'user story' Push cloud monitoring alerts to mobile devices
```

Create a work item from message actions

Often, discussions in a channel call for creation of work items. You can use message actions to create a work item. The selected message is prefilled in the description section of the work item. A link back to the conversation in the channel is stored in the Discussion section of the newly created work item, giving users access to the discussion that led to the creation of the work item.

- To create work items using message actions



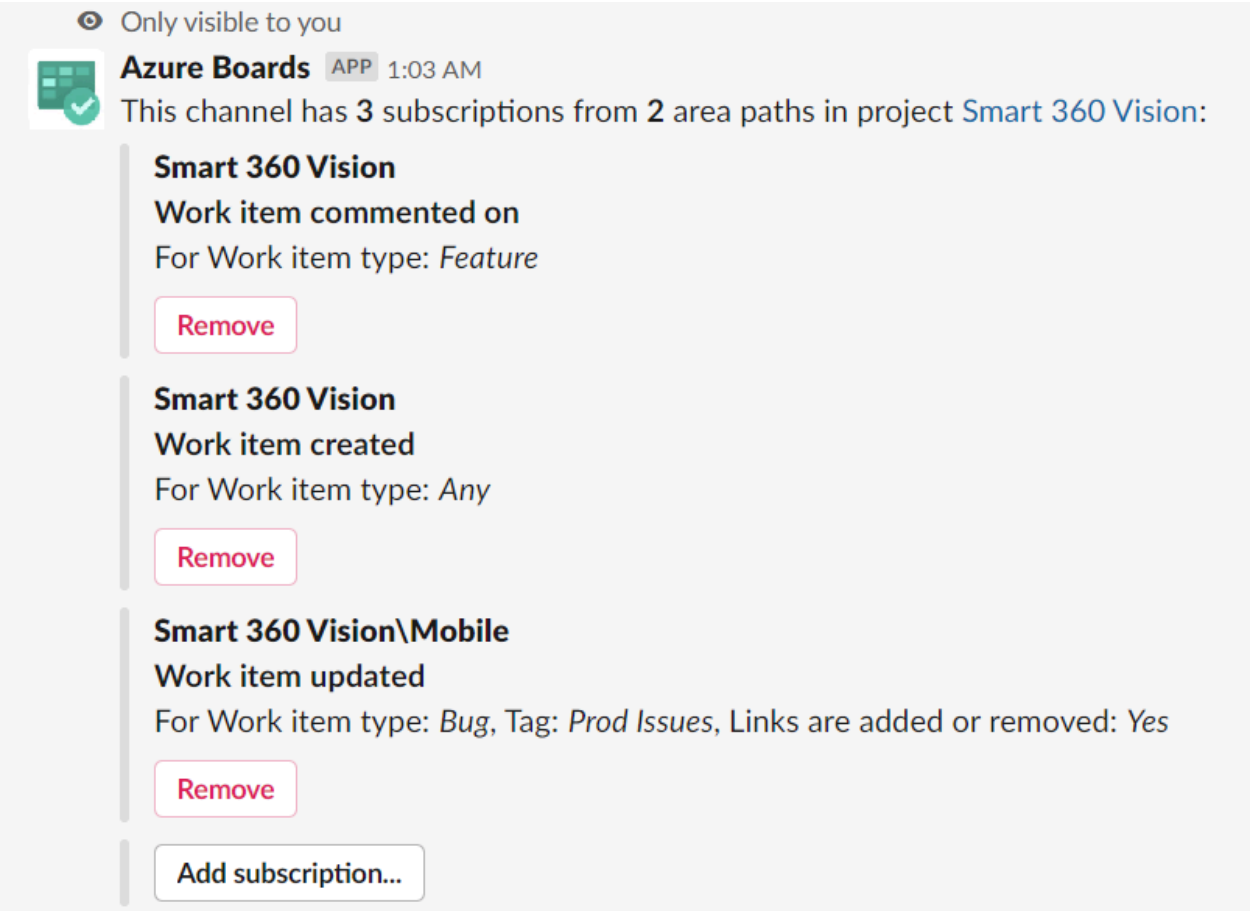
Manage Azure Boards subscriptions

1. To view, add and remove subscriptions for a channel, use the `/azboards subscriptions` command:

```
/azboards subscriptions
```

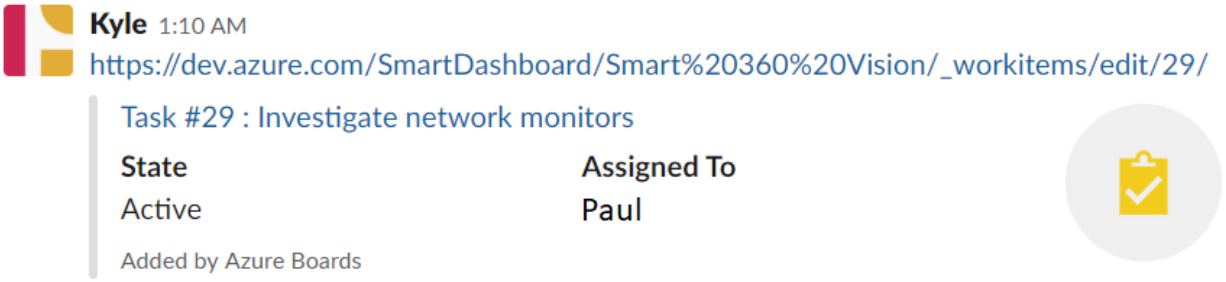
   This command lists all the current subscriptions for the channel and allows you to add new subscriptions and remove existing ones. As part of adding subscriptions, you can also customize what you get notified on by using various filters.

[!NOTE]Team administrators aren't able to remove or modify subscriptions created by Project administrators.



Previews of work item URLs

To support collaboration around work items discussed within a channel, a preview of work items referenced in the channel is displayed. When a user pastes the work item URL, a preview is shown similar to the following image. This preview helps to keep work-item-related conversations relevant and correct.



For this feature to work, users have to be signed-in. Once they're signed in, this feature works for all channels in a workspace.

Unlink a project from a channel

A Slack channel can only link to one Azure Boards project at a time. To link to a different project, you must first unlink the current project using `/azboards unlink` command.

Unlinking a project deletes all the subscriptions along with added area paths from the channel. If the channel has no subscriptions, any user can unlink a project. However if a channel has subscriptions, only project admins can unlink a project from a channel.

Command reference

The following table lists all the `/azboards` commands you can use in your Slack channel.

Expand table

Slash command	Functionality
/azboards link [project url]	Link a project to this channel to create work items and receive notifications
/azboards subscriptions	Add or remove subscriptions for this channel
/azboards create or /azboards create [work item type] [title]	Create a work item
/azboards addAreapath [area path]	Add an area path from your project to this channel
/azboards signin	Sign in to your Azure Boards organization
/azboards signout	Sign out from your Azure Boards organization
/azboards unlink	Unlink a project from this channel
/azboards feedback	Report a problem or suggest a feature

Manage work in private channels

The Azure Boards app for Slack can help you create work items and monitor the work item activity in your private channels as well. To invite the bot to your private channel, enter `/invite @azboards`. After you post that, you can create work items and manage your notifications in the same way as you would for a public channel.

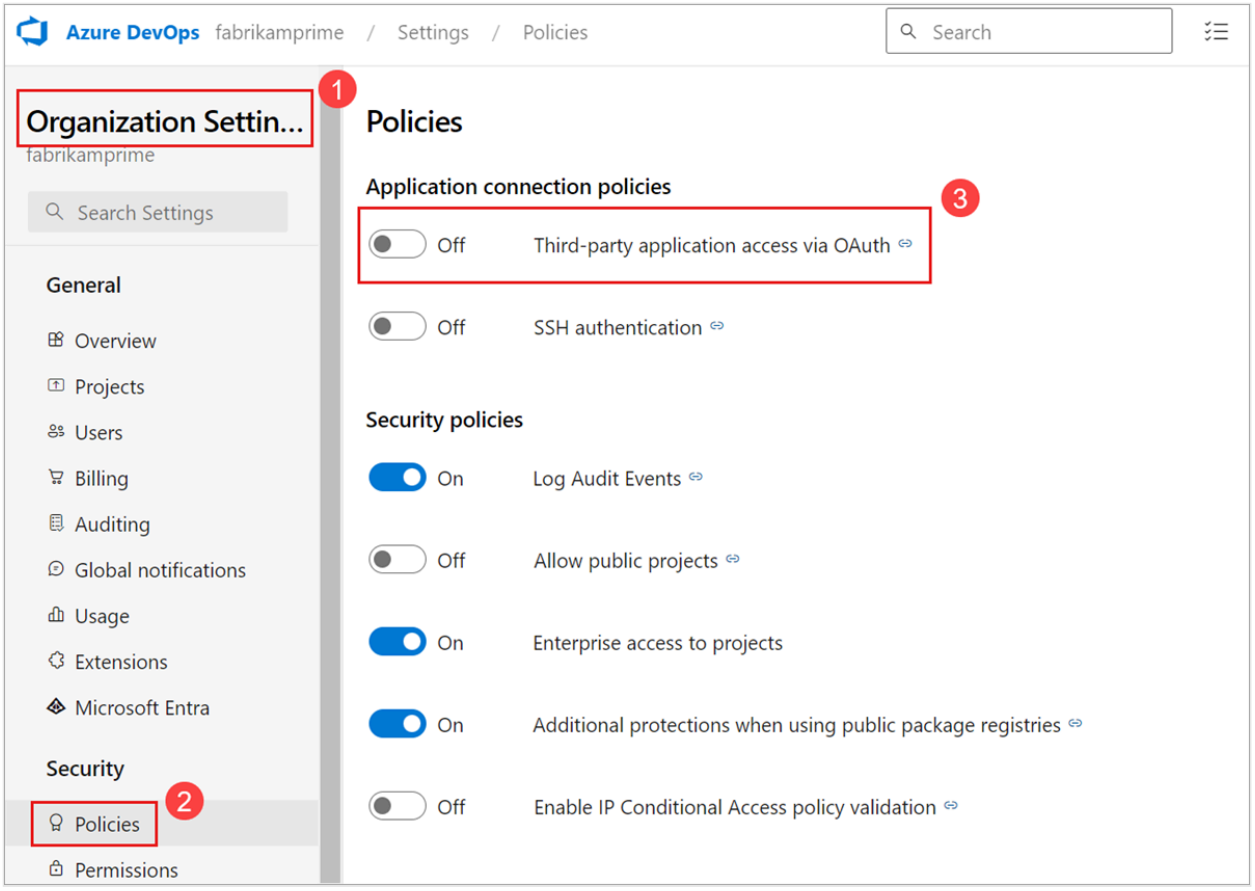
Troubleshoot errors

If you're experiencing the following errors when using the Azure Boards App for Slack, follow the procedures in this section.

- Sorry, something went wrong. Please try again.
- Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

Sorry, something went wrong. Please try again.

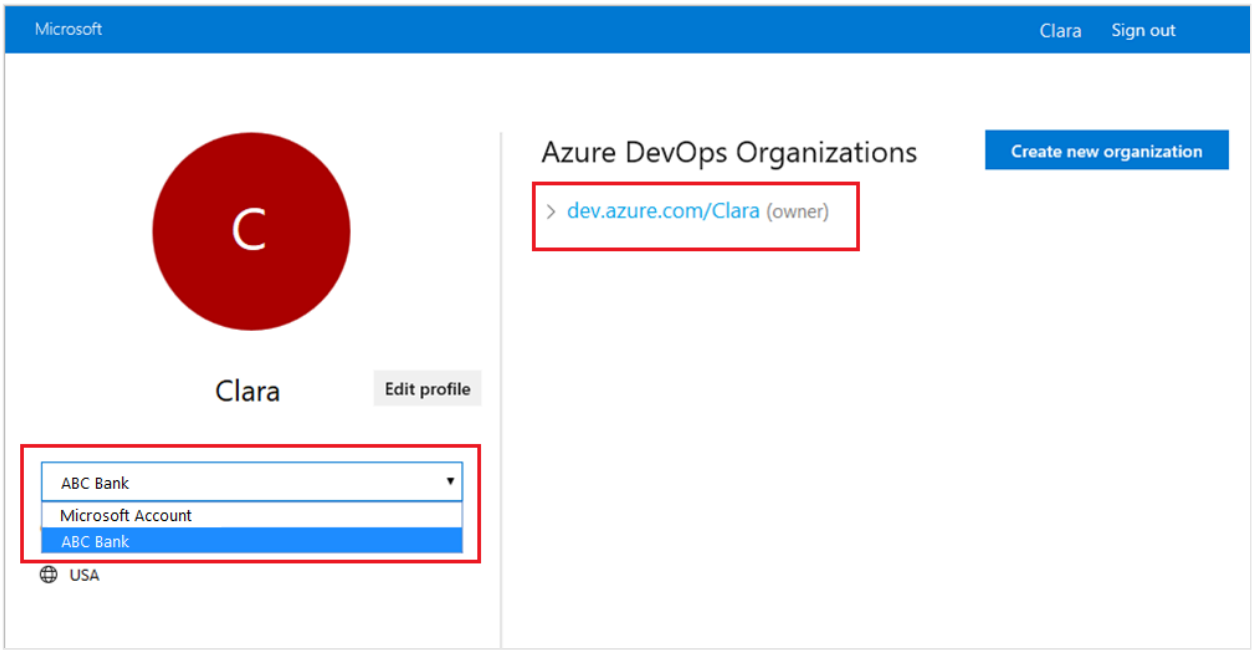
The Azure Boards app uses the OAuth authentication protocol, and requires Third-party application access via OAuth for the organization to be enabled. To enable this setting, go to **Organization settings** > **Security** > **Policies**, and set the **Third-party application access via OAuth for the organization** setting to **On**.



Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

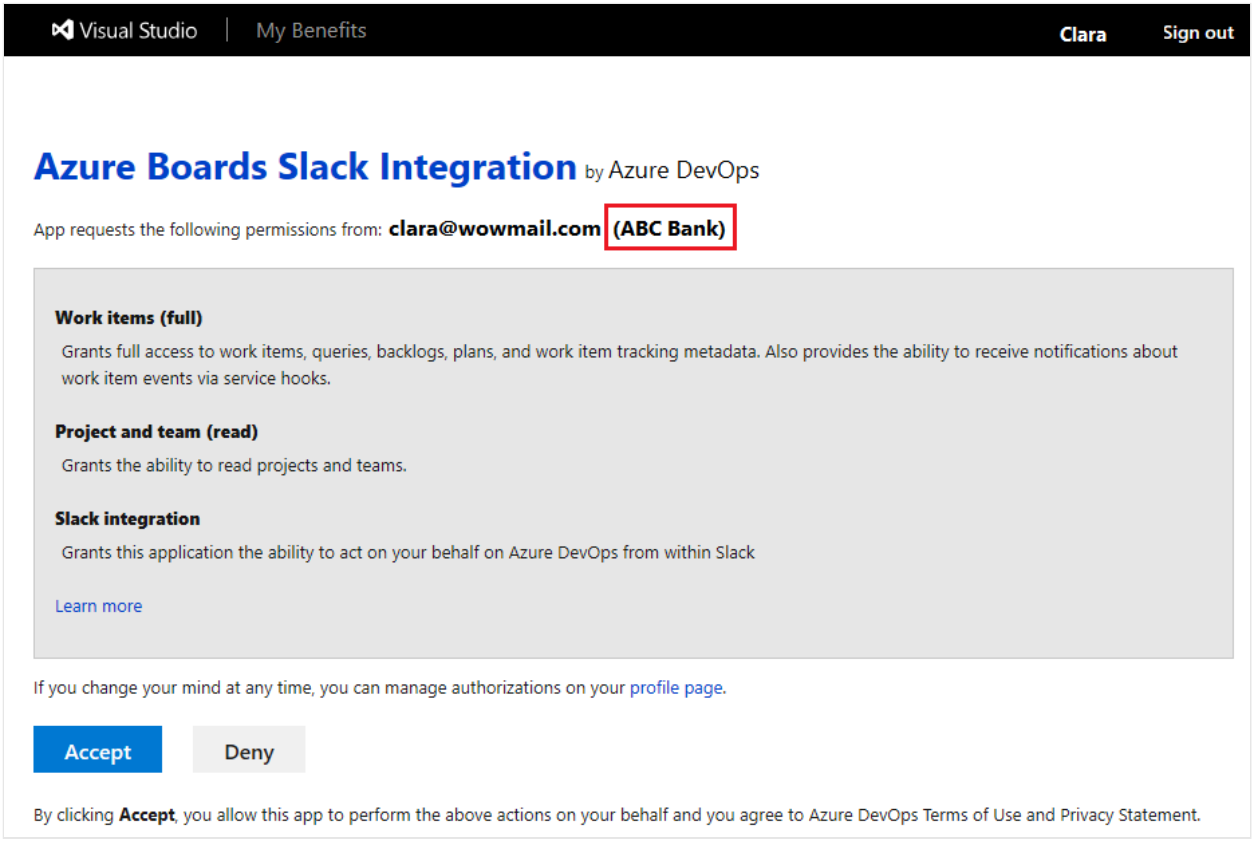
Sign out of Azure DevOps by going to `https://aka.ms/VsSignout` using your browser.

Open an **In private** or **incognito** browser window and go to `https://aex.dev.azure.com/me` and sign in. In the dropdown under the profile icon to the left, select the directory that contains the organization containing the project that you want to link.



In the **same browser**, start a new tab, go to `https://slack.com`, and sign in to your work space (**use web client**). Run the `/azboards signout` command followed by the `/azboards signin` command.

Select the `Sign in` button and you get redirected to a consent page like the one in the following example. Ensure that the directory shown beside the email is same as what was chosen in the previous step. Accept and complete the sign-in process.



If these steps don't resolve your authentication issue, reach out to us at Developer Community.

Related articles

- Define area paths and assign to a team
- Use Azure Pipelines with Slack
- Use Azure Repos with Slack
- Create a service hook for Azure DevOps with Slack

Feedback

Was this page helpful? Yes No

Provide product feedback

Use Azure Boards in Microsoft Teams

Article • 01/30/2025

Azure DevOps Services

This article describes how to use the Azure Boards and Azure DevOps apps for Microsoft Teams to create and monitor Azure Boards work items from your Teams channels.

> **Note**
>
> This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. For more information, see Azure DevOps Feature Timeline.

By using the Azure Boards app for Microsoft Teams, you can:

- Set up subscriptions to create and manage work items and work item events in your Teams channels.
- Create work items directly from channel conversations.
- Search for and share work items across channels using the messaging extension.
- View previews of work items from URLs.

> **Note**
>
> - Azure Boards notifications aren't supported inside Teams chat or direct messages.
> - The Azure Boards app for Microsoft Teams isn't supported for O365 Government Community Cloud (GCC) customers that use an Azure Commercial subscription in conjunction with a GCC tenant.

Prerequisites

Expand table

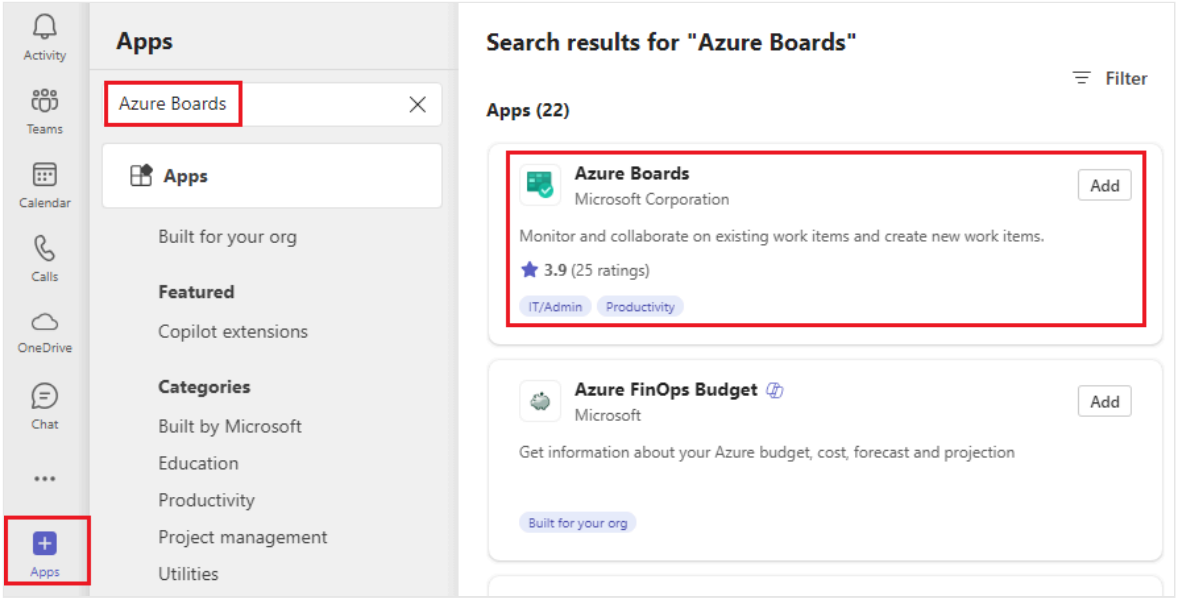
Category	Requirements
Access levels	- **Contributor** for the project. - Access to a team and channel in Microsoft Teams where you can install an app.

Category	Requirements
Permissions	To create subscriptions in a Slack channel for work item events: Member of the Project Administrators or Team Administrators group.
Policies	To receive notifications: **Third party application access via OAuth** policy enabled for the organization.

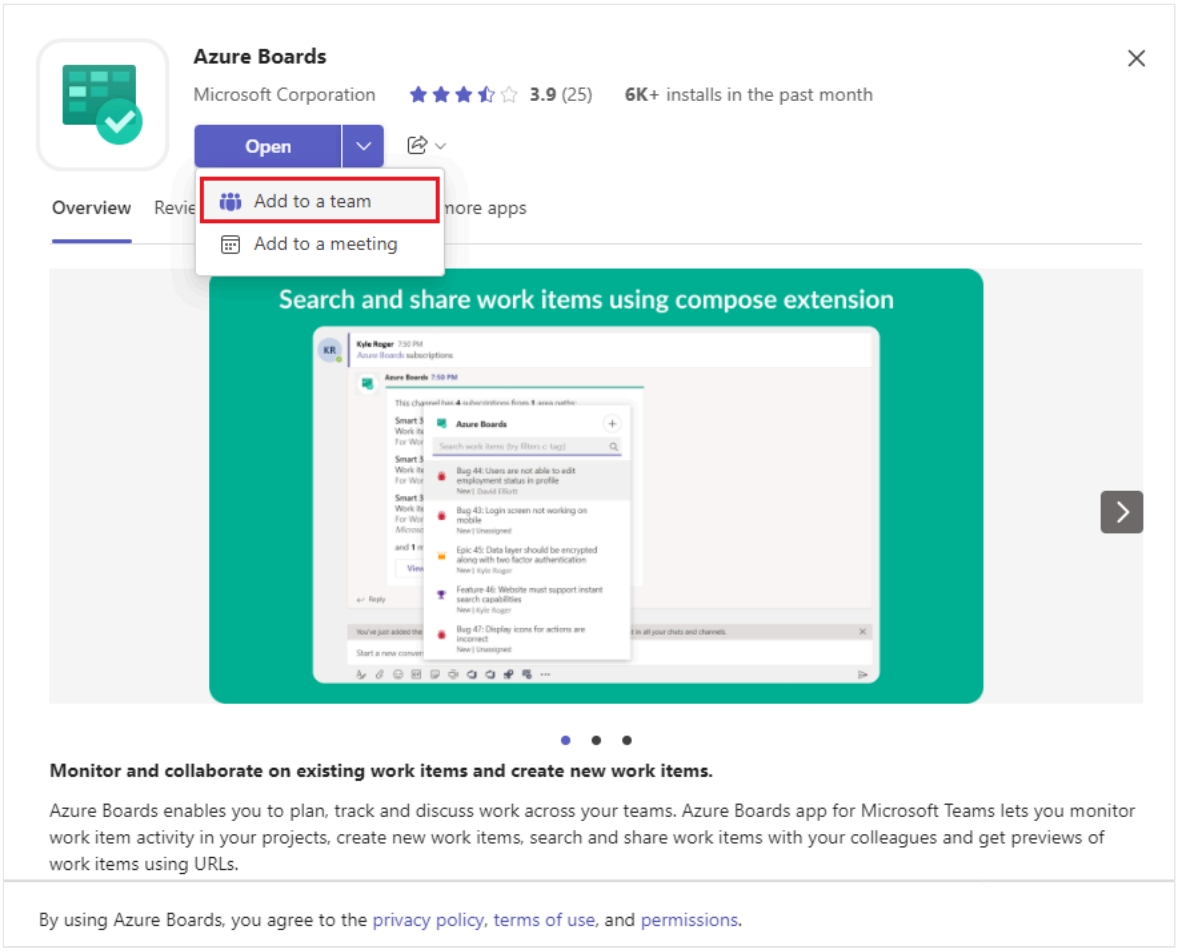
Add the Azure Boards app to Teams

To add the Azure Boards app to your Teams channels, do the following steps.

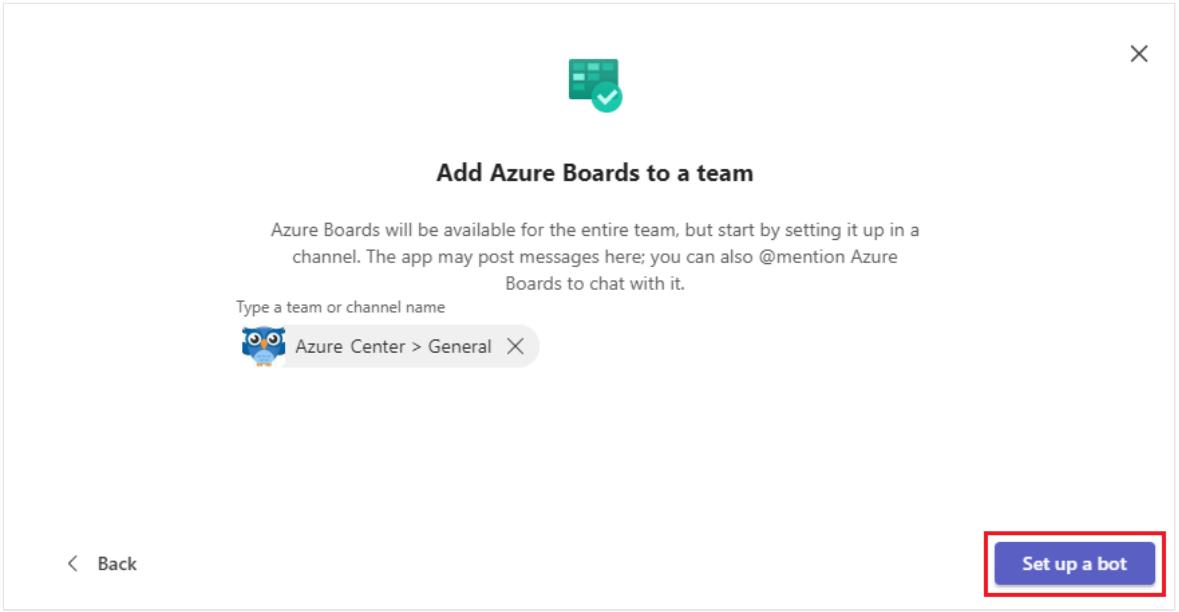
1. In Microsoft Teams, select **Apps**, search for *Azure Boards*, and then select **Azure Boards**.



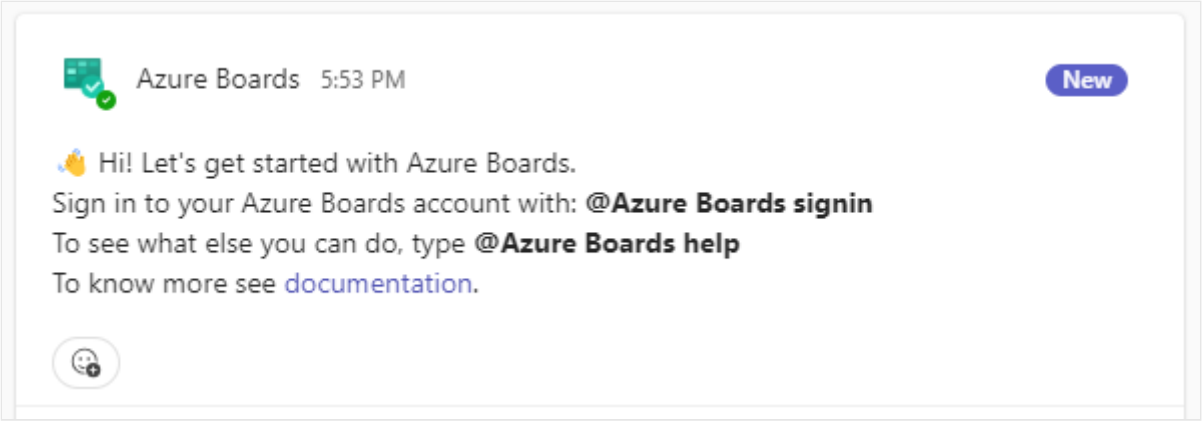
2. Select the dropdown arrow next to **Open**, and select **Add to a team**.



3. Select or enter your team name, and then select **Set up a bot**.



After the app installs, you see a welcome message in the Teams conversation pane.



Azure Pipelines app commands

You can use the following `@azure boards` commands to interact with the Azure Boards app in your Teams channel.

Expand table

Command	Functionality
@azure boards sign in	Sign in to your Azure Boards organization.
@azure boards sign out	Sign out from your Azure Boards organization.
@azure boards link <project url>	Link a project to this channel to create work items and receive notifications.
@azure boards unlink	Unlink a project from this channel.
@azure boards subscriptions	Add or remove subscriptions for this channel.
@azure boards addAreapath <area path>	Add an area path from your project to this channel.
@azure boards feedback	Report a problem or suggest a feature.
@azure boards help	Get help on the commands.

Link your Azure Boards project to the app

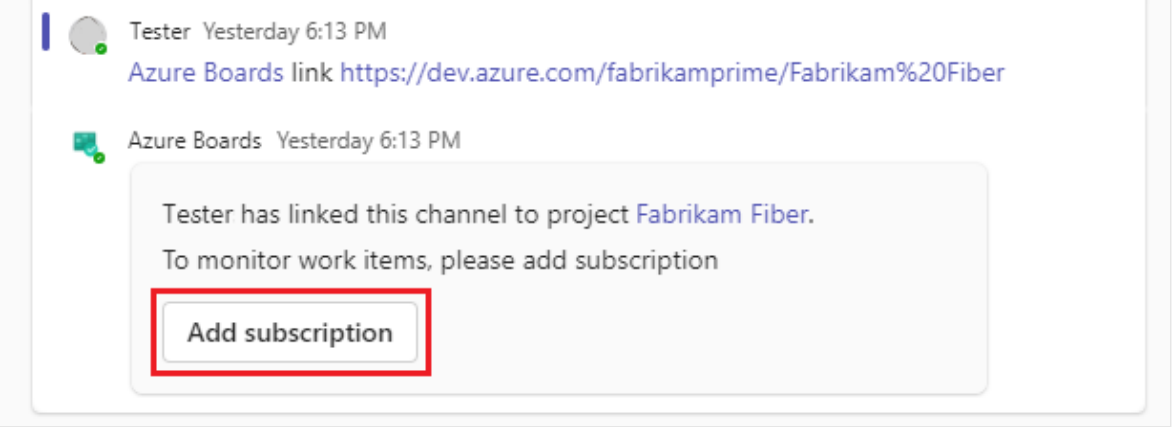
To use the app, sign in to Azure Boards and link your Azure Boards project to your Teams channel.

1. In the Teams conversation pane, enter `@azure boards signin`.
2. Select **Sign in** and complete authentication to Azure Boards. Select **Sign in with different email** if your Microsoft Teams and Azure Boards are in different tenants.

3. Use the `@azure boards link` command to link to your Azure DevOps project URL. For example:

```
@azure boards link https://dev.azure.com/myorg/myproject/
```

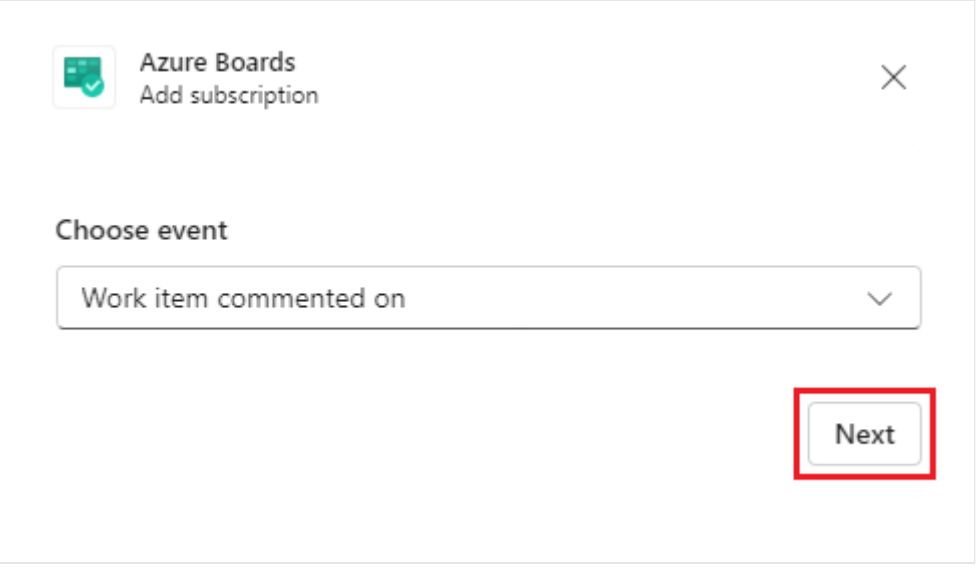
4. Select the **Add subscription** button in the linked project notification to start monitoring your project.



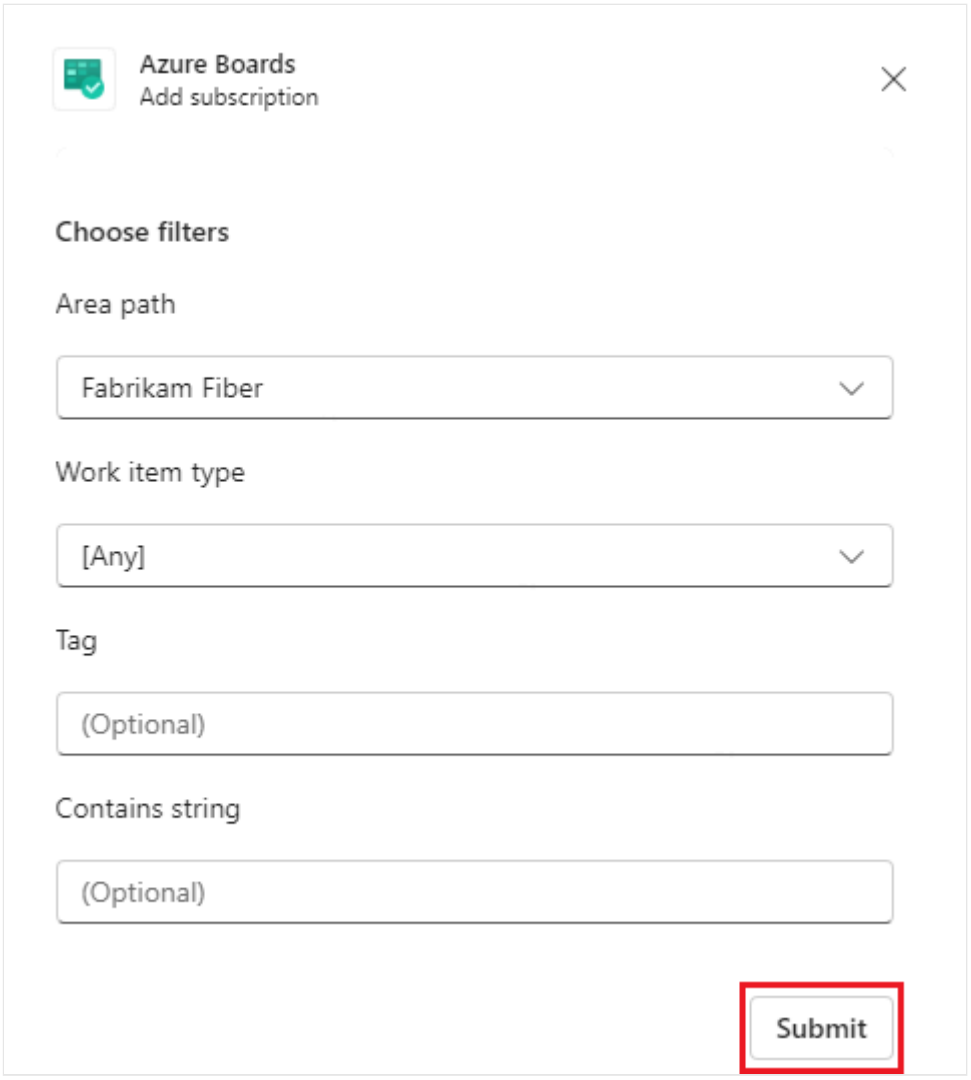
Set up subscriptions

After your project is linked, begin monitoring project work items by selecting the **Add subscription** button. You can add more subscriptions anytime by using the `@azure boards subscriptions` command.

1. Under **Choose event**, select the event you want to subscribe to and select **Next**.



2. Under **Choose filters**, select the **Area path**, **Work item type**, and optionally specify **Tags** or a specific string to filter on, and then select **Submit**.



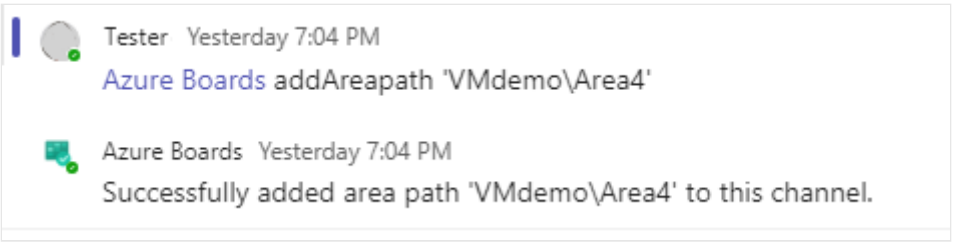
Add area paths

Area paths that have subscriptions in the channel, recently accessed area paths, and area paths that you add by using the `@azure boards addAreapath` command appear in the **Area path** dropdown menu when you create a subscription. If the area path that your team works on doesn't appear in the dropdown menu, you can add it so that it's always available for creating work items and subscriptions. This feature is especially useful for teams with more than 100 area paths in their project.

Use the `@azure boards addAreapath` command to add area paths from your project to the Teams channel. For example:

```
@azure boards addAreapath 'VMdemo\Area4'
```

You get a success message.

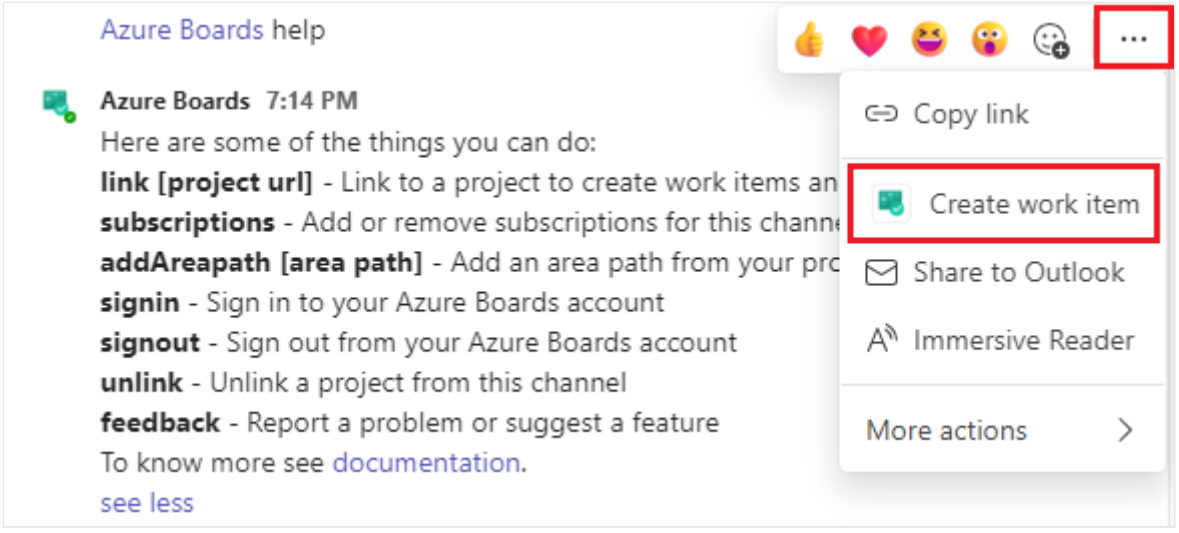


If you choose the project name as your area path, you get notifications for all the area paths in the project.

Create work items

You can use the Azure Boards app to create work items from your channel by using a message action.

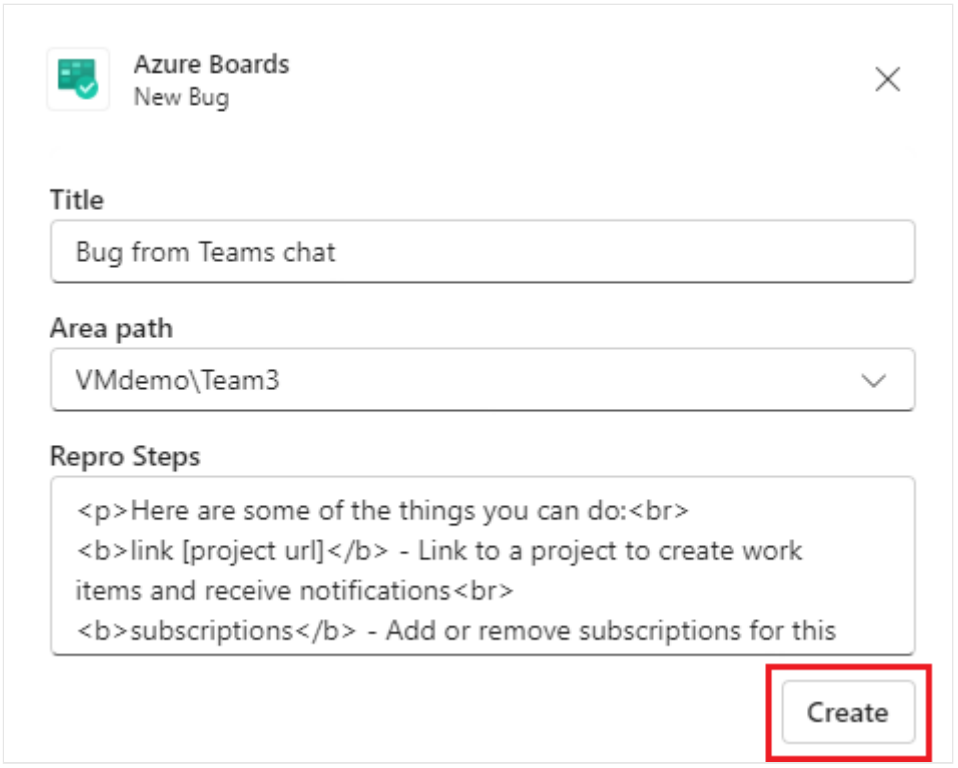
1. In any message in the channel, select the **More actions** ellipse in the actions panel, and then select **Create work item**.



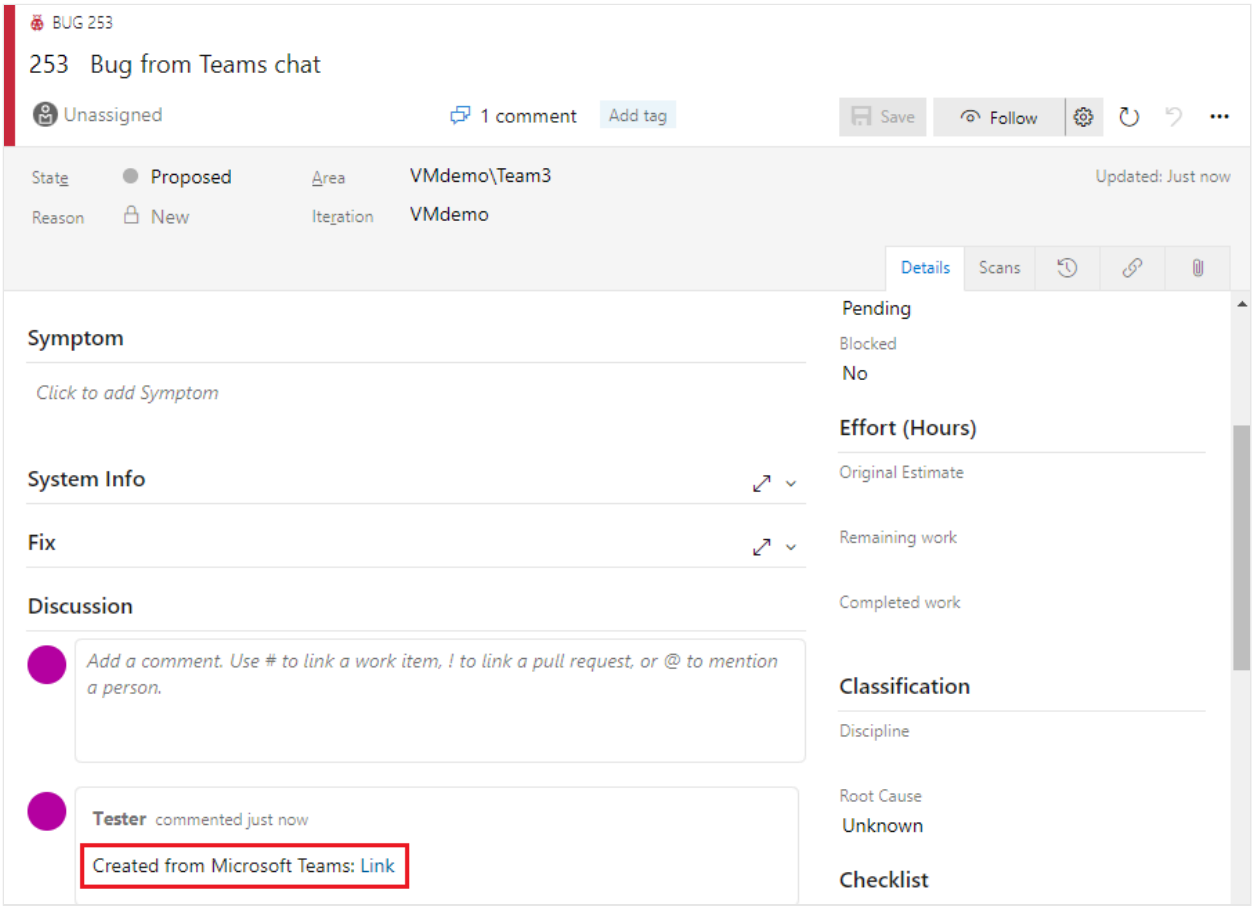
2. Select the type of work item you want to create, and select **Next**.

3. Enter a **Title** and select an **Area path** for the work item.

4. The text of the message becomes the work item **Description** or **Repro Steps**, depending on work item type, or you can edit this text. Select **Create**.

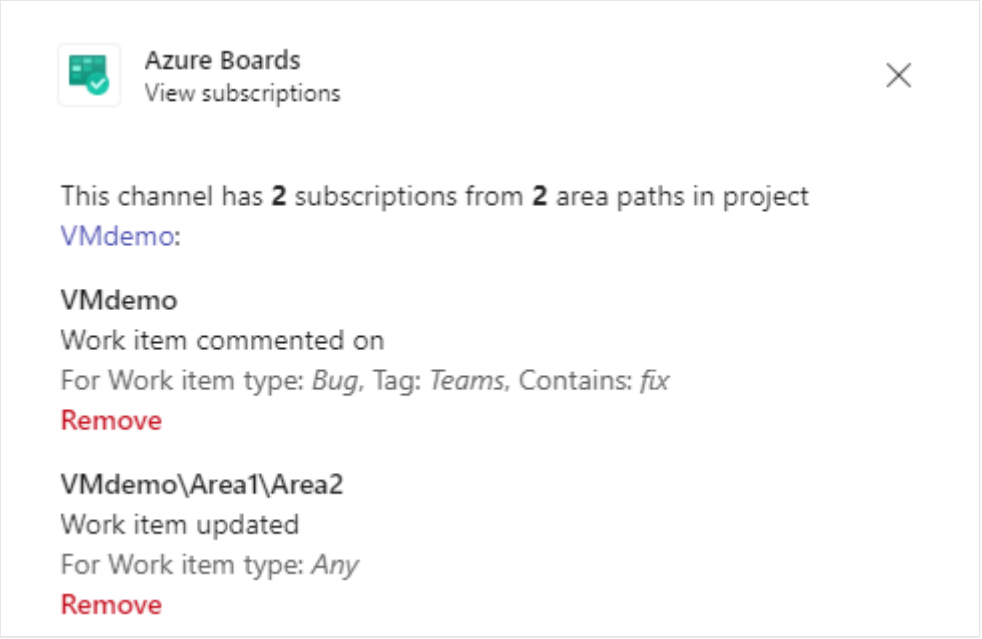


The new work item appears in Azure Boards and contains a link back to the Teams item that generated the work item.



Manage Azure Boards subscriptions

To view, add, and remove subscriptions for a channel, use the `@azure boards subscriptions` command. This command lists all the current subscriptions for the channel. You can add new subscriptions and remove existing ones. When you add a subscription, you can customize notifications by using filters.

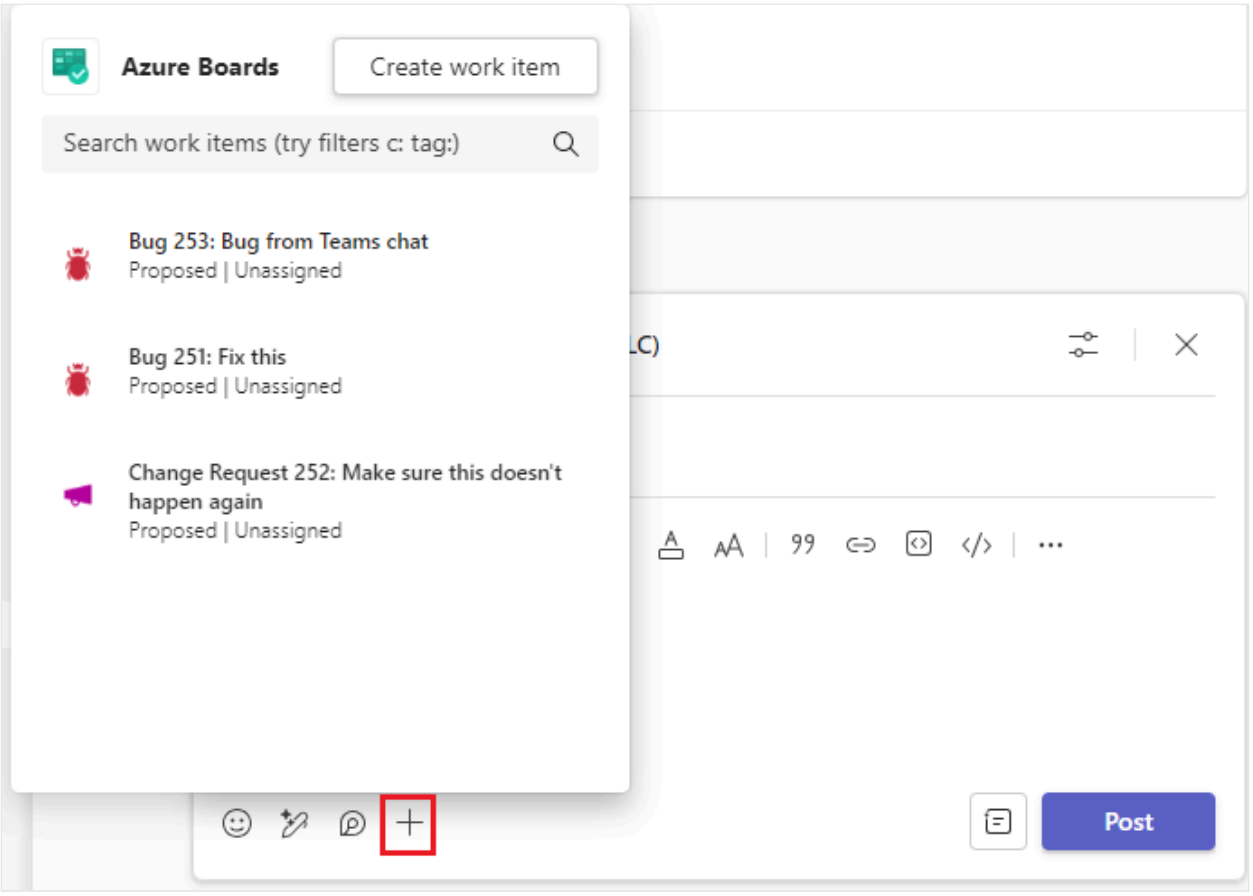


> **Note**
>
> Team Administrators can't remove or modify subscriptions that Project Administrators created.

Use the compose extension

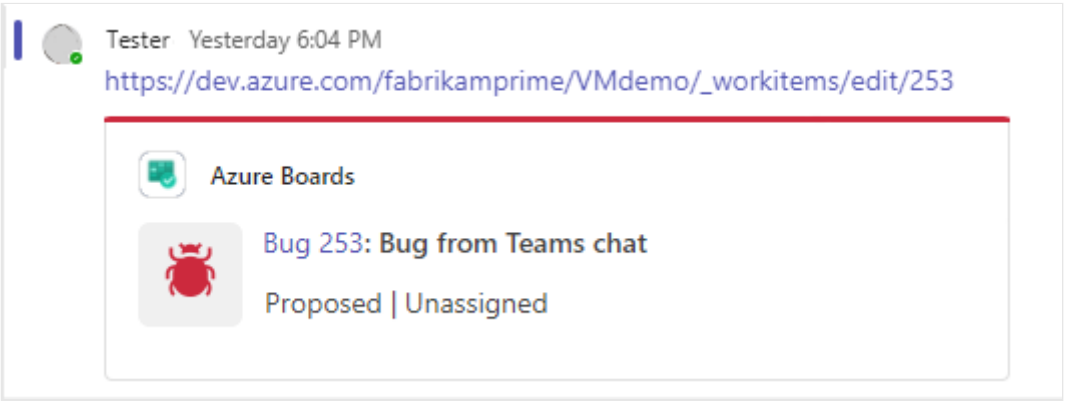
To help you search for and share work items, the Azure Boards app for Microsoft Teams supports the compose extension. You can search for work items by work item ID, title, or supported functional command. For a list of supported commands, see Functional work item search.

To use the compose extension, be signed in to the Azure Boards project in the Teams channel. Select the + symbol in the message field, select **Azure Boards**, and then search for a work item. You can also select **Create work item** to create a new work item.



Preview work items

To support collaboration around work items you discuss in a channel, the Azure Boards app displays a preview of work items you reference. When you paste in a work item URL or select a work item from the compose extension, the app shows a preview similar to the following image. This URL unfurling feature works for all channels in the team.



Unlink a project from a channel

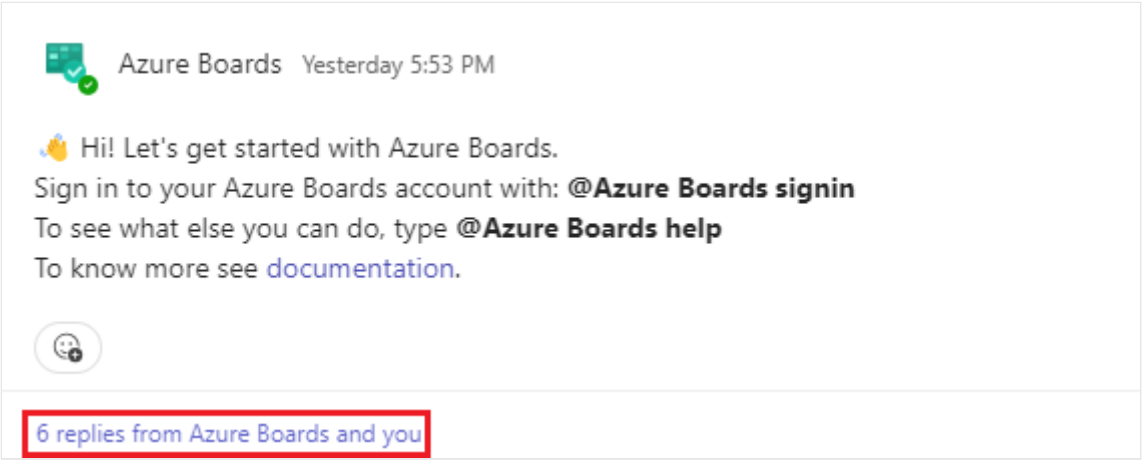
A Teams channel can link to only one Azure Boards project at a time. To link to a different project, you must first unlink the current project by using `@azure boards unlink`.

Unlinking a project deletes all the project subscriptions and added area paths from the channel. If the channel has no subscriptions, any user can unlink a project. If the channel has subscriptions, only Project Administrators can unlink the project.

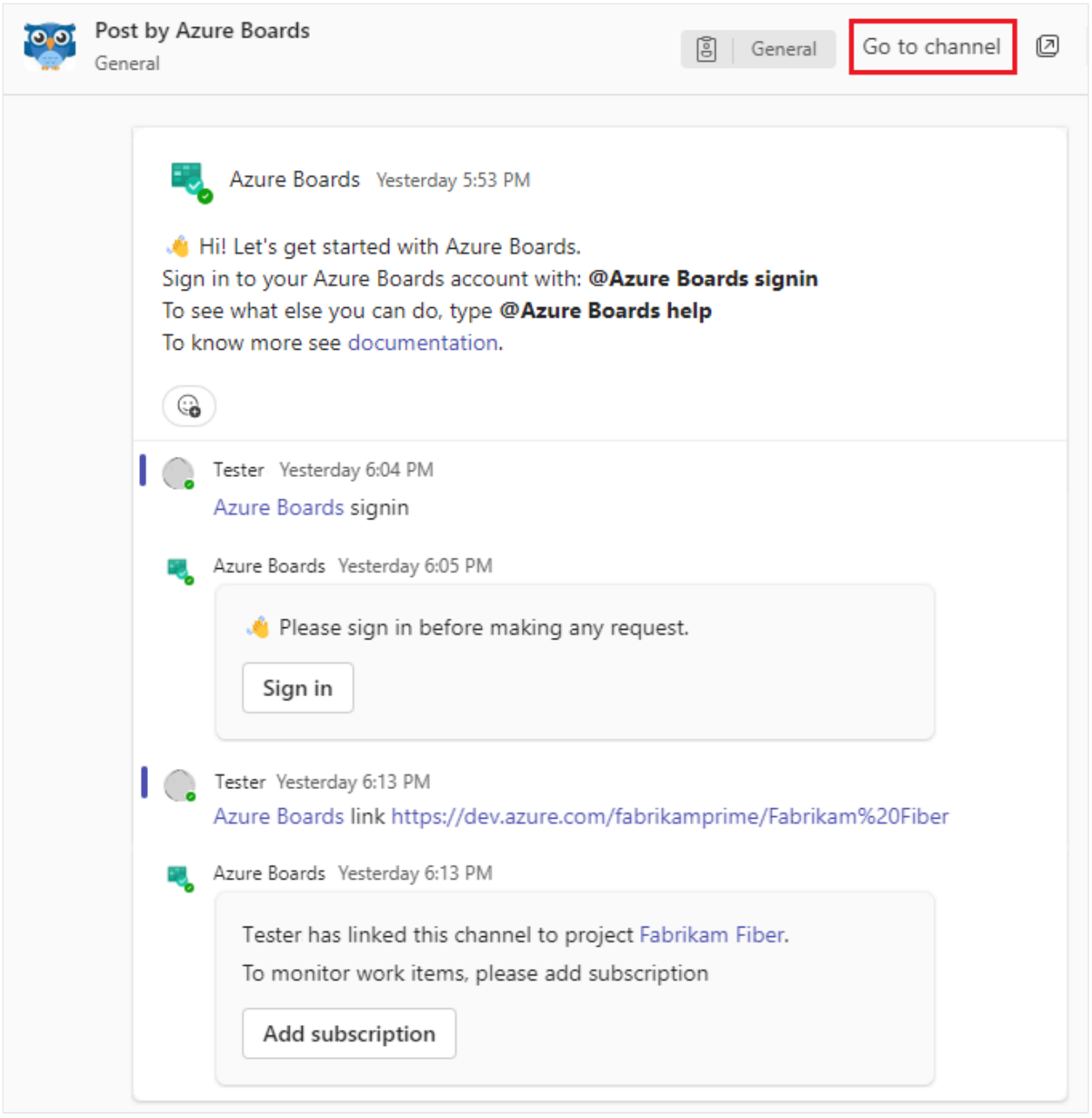
Expand and collapse threads

A Teams channel collapses posts in threads to logically link and reduce related posts in the channel. All replies to a particular post are linked together.

To expand the thread, select the compacted thread link.



To return to the channel and collapse the thread, select **Go to channel**.

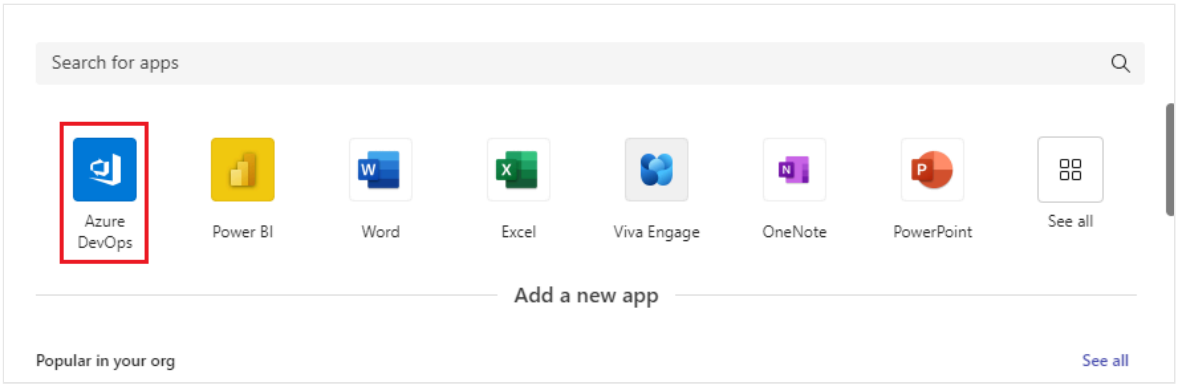


Configure Azure DevOps tabs

To bring your project dashboard or Kanban board into Teams, you can install the Azure DevOps app in a tab in your Teams channel. The Azure DevOps app lets you insert content from the app in messages, and get notifications from the app in your channels.

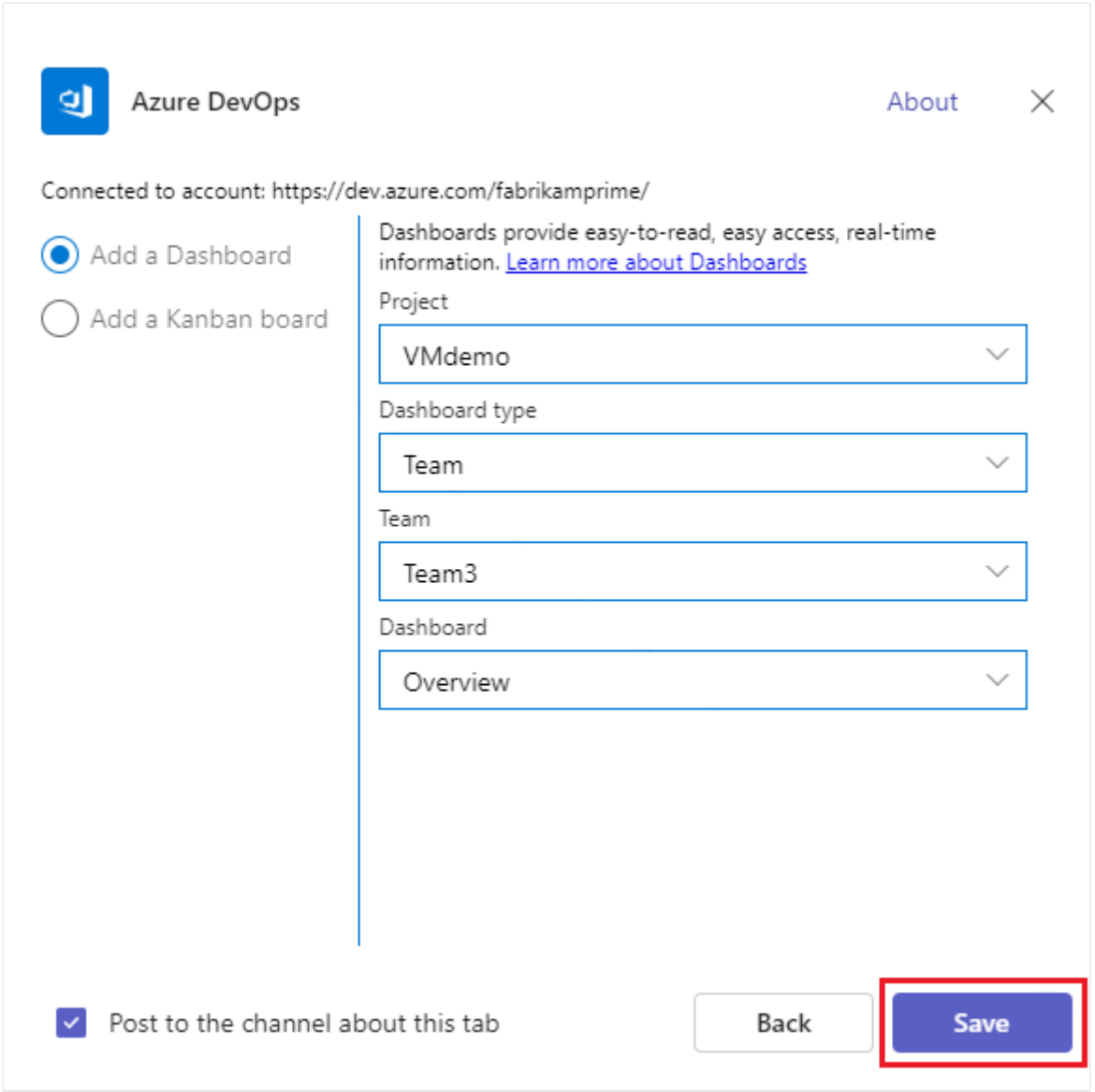
1. In Teams, select the + symbol on the top navigation of your channel or select **Apps** from the left menu.

2. Search for if necessary, and then select **Azure DevOps**.



3. Select and sign in to your Azure DevOps organization.

4. On the **Azure DevOps** screen, select a **Project**, and whether to add a **Dashboard** or a **Kanban board** to the tab. Select other configurations depending on your choice, and select whether you want to post to the channel about adding the tab.



5. Select **Save**. The new tab and board appear in your channel.

Connect different tenants

If you use different emails or tenants for Microsoft Teams and Azure DevOps, follow these steps to sign in and connect based on your settings.

Expand table

Microsoft Teams	Azure DevOps	Sign in action
email1@abc.com (tenant1)	email1@abc.com (tenant1)	Select **Sign in**
email1@abc.com (tenant1)	email2@abc.com (tenant2)	1. Sign in to Azure DevOps. 2. In the same browser, start a new tab and go to https://teams.microsoft.com/. 3. Run the `sign in` command and select **Sign in**.
email1@abc.com (tenant1)	email2@pqr.com (tenant2)	1. Select **Sign in with different email address**. 2. In the email ID picker, use the email2 to sign in.
email1@abc.com (tenant1)	email2@pqr.com (nondefault tenant3)	Not supported.

Troubleshoot authentication issues

If you receive the error **Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions**, try the following steps to resolve the error.

1. In the same browser, start a new tab and sign in to `https://teams.microsoft.com/`.
2. In this tab, go to the channel where the Azure Boards app for Microsoft Teams is installed and run the `@azure boards signout` command and then the `@azure boards signin` command.
3. Select the **Sign in** button, and complete the sign-in process. Ensure that the directory shown is the same one you chose in the previous step.

If these steps don't resolve your authentication issue, reach out to the Developer Community.

Related articles

- Define area paths and assign to a team
- Use Azure Pipelines with Microsoft Teams

- Use Azure Repos with Microsoft Teams

Feedback

Was this page helpful? Yes No

Provide product feedback

Azure Repos with Slack

Article • 04/30/2024

Azure DevOps Services

If you use Slack, you can use the Azure Repos app for Slack to easily monitor your Azure Repos repositories. Set up and manage subscriptions to receive notifications in your channel whenever code is pushed or checked in and whenever a pull request (PR) gets created, updated, or merged. This app supports both Git and Team Foundation Version Control (TFVC) events.

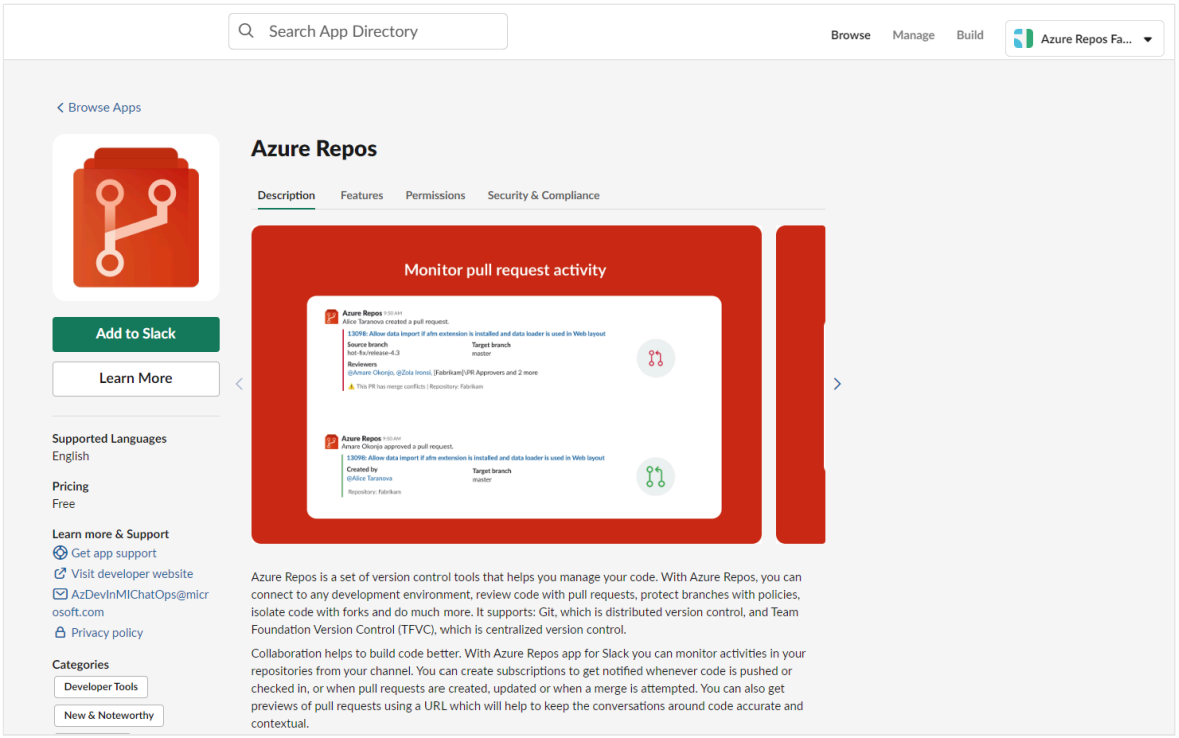
Prerequisites

- To create subscriptions in a Slack channel for repository-related events, you must be a member of the **Project Administrators** group, or be a team administrator. To get added, see Change project-level permissions or Add a team administrator.
- To receive notifications, the **Third-party application access via OAuth** setting must be enabled for the organization. For more information, see Change application access policies for your organization.

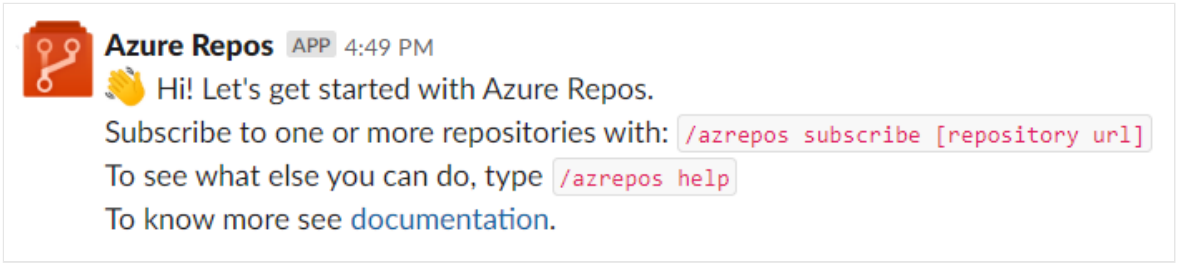
> **Note**
>
> - You can only link the Azure Repos app for Slack to a project hosted on Azure DevOps Services at this time.
> - Notifications are currently not supported inside direct messages.

Add the Azure Repos app to your Slack workspace

1. Go to the Azure Repos Slack app and select **Add to Slack**.



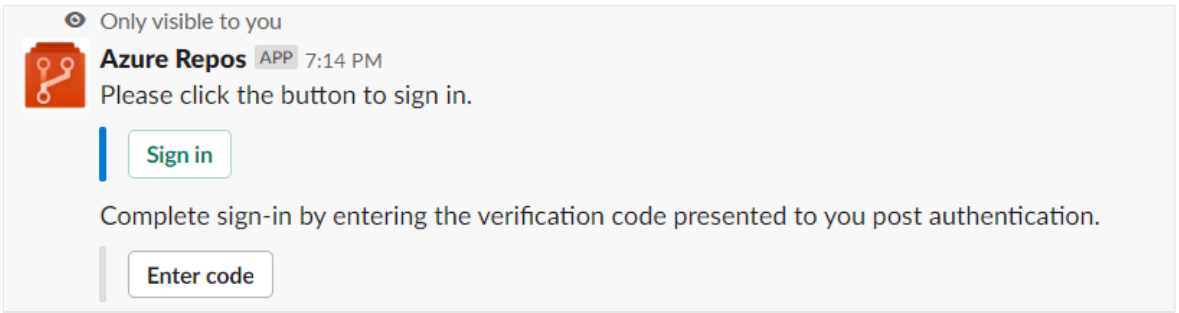
Once added, you receive something like the following welcome message.



2. Use the `/azrepos` Slack handle to interact with the app. A full list of commands is provided in the Command reference section of this article.

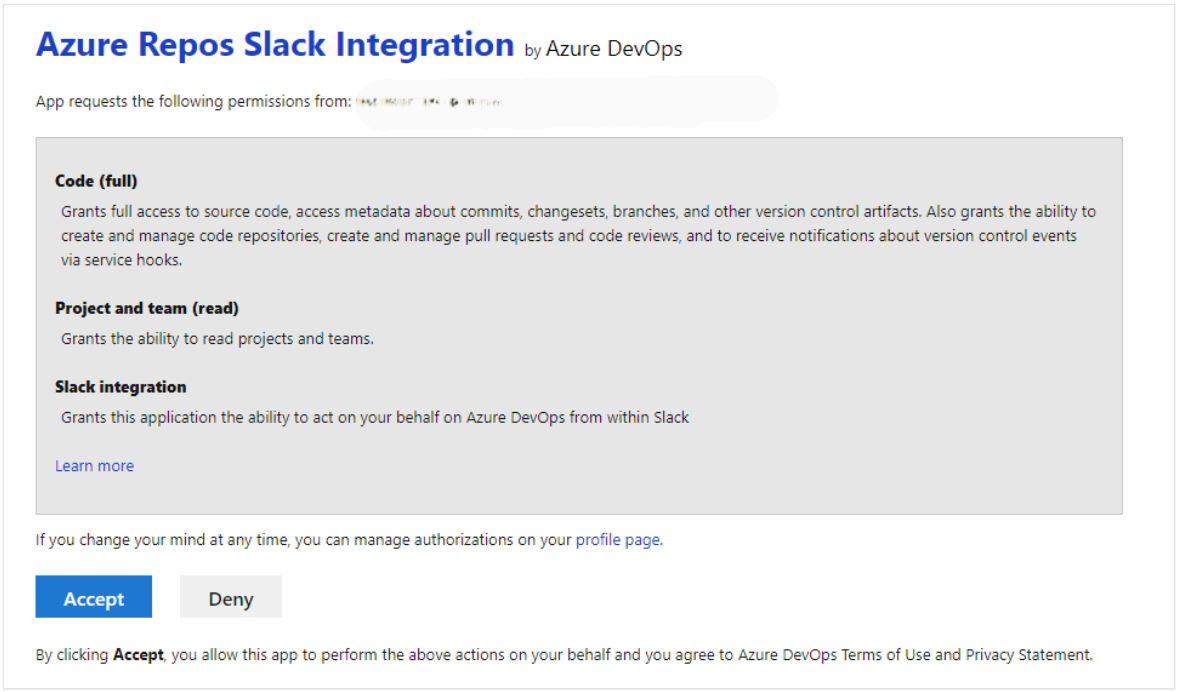
Connect the Azure Repos app to your repositories

1. Connect and authenticate yourself to Azure Repos using `/azrepos signin` command.



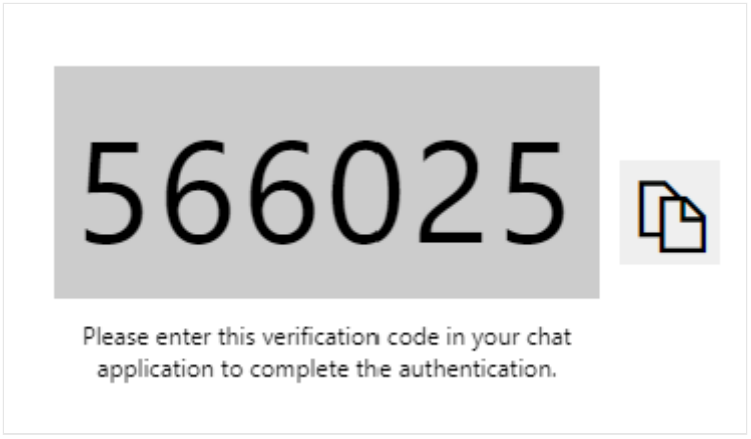
2. Select **Sign in**.

3. **Accept** the Azure Repos Slack Integration.

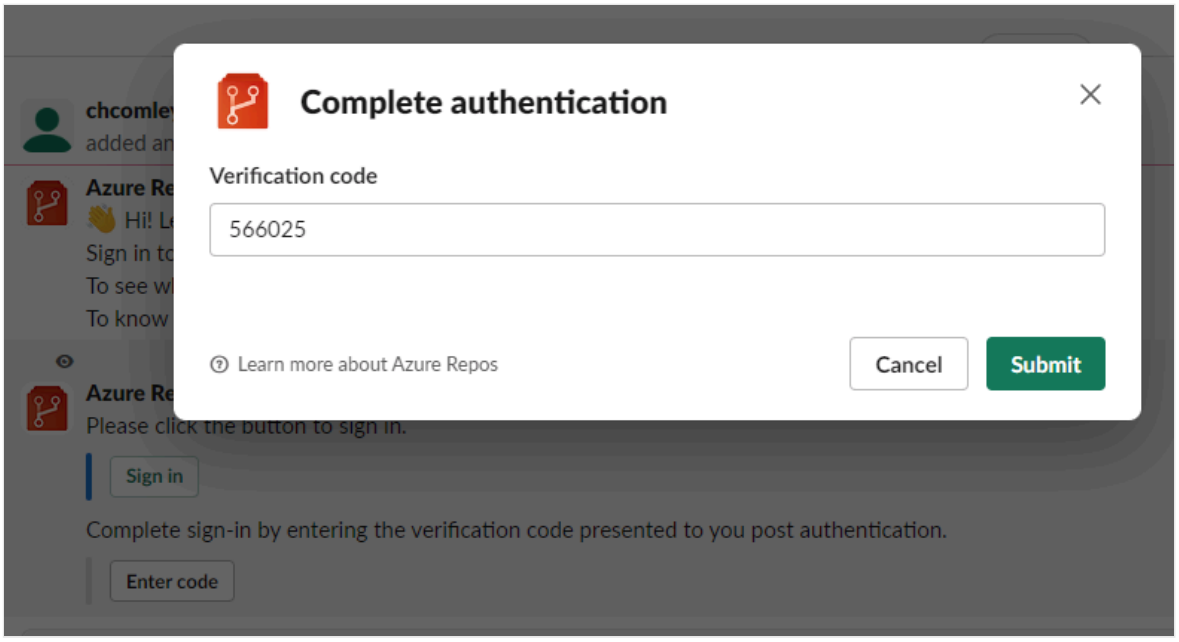


A verification code displays for use in your chat app to complete authentication.

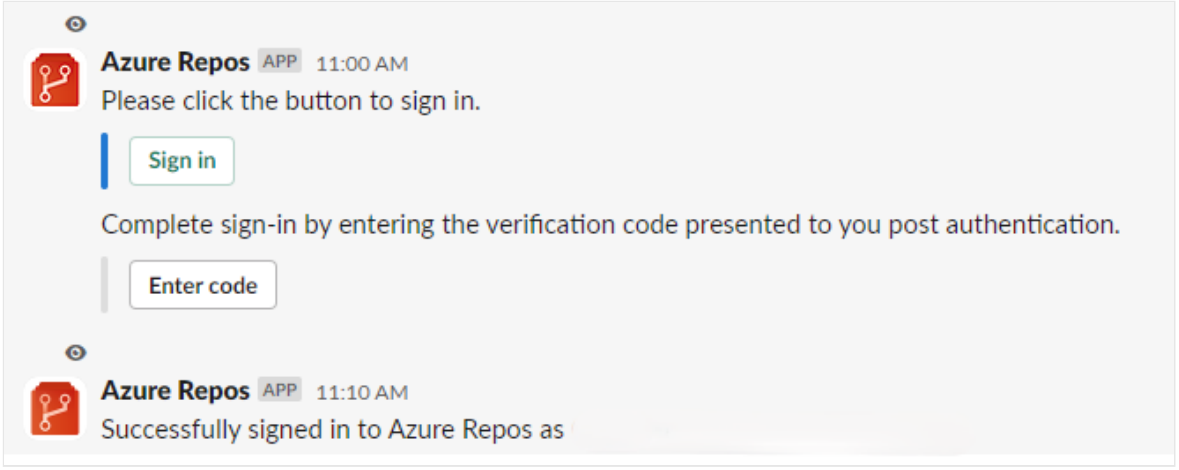
4. Copy the authentication code.



5. Select **Enter code**, paste the code, and then select **Submit**.



Confirmation of sign-in displays within the chat.



To start monitoring all Git repositories in a project, use the following slash command inside a channel:

```
slash
```

```
/azrepos subscribe [project url]
```

The project URL can be to any page within your project (except URLs to repositories). For example:

```
slash
```

```
/azrepos subscribe https://dev.azure.com/myorg/myproject/
```

You can also monitor a specific repository using the following command:

```
slash
```

```
/azrepos subscribe [repository url]
```

The repository URL can be to any page within your repository that has your repository name. For example, for Git repositories, use:

slash

```
/azrepos subscribe https://dev.azure.com/myorg/myproject/_git/myrepository
```

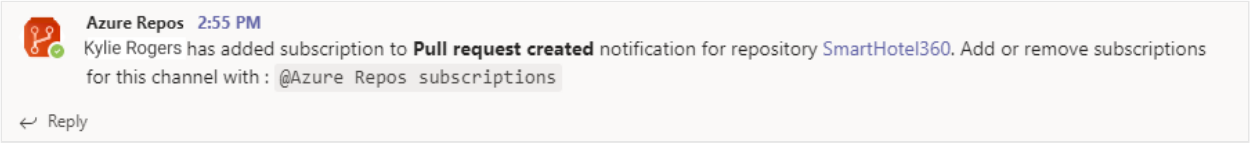
For TFVC repositories, use:

slash

```
/azrepos subscribe https://dev.azure.com/myorg/myproject/_versionControl
```

> **Note**
>
> You can only subscribe to public repositories.

The subscribe command gets you started with a default subscription. For Git repositories, the channel is subscribed to the **Pull request created** event (with target branch = main), and for TFVC repositories, the channel is subscribed to the **Code checked in** event.



Manage subscriptions

To view, add, or remove subscriptions for a channel, use the following `subscriptions` command:

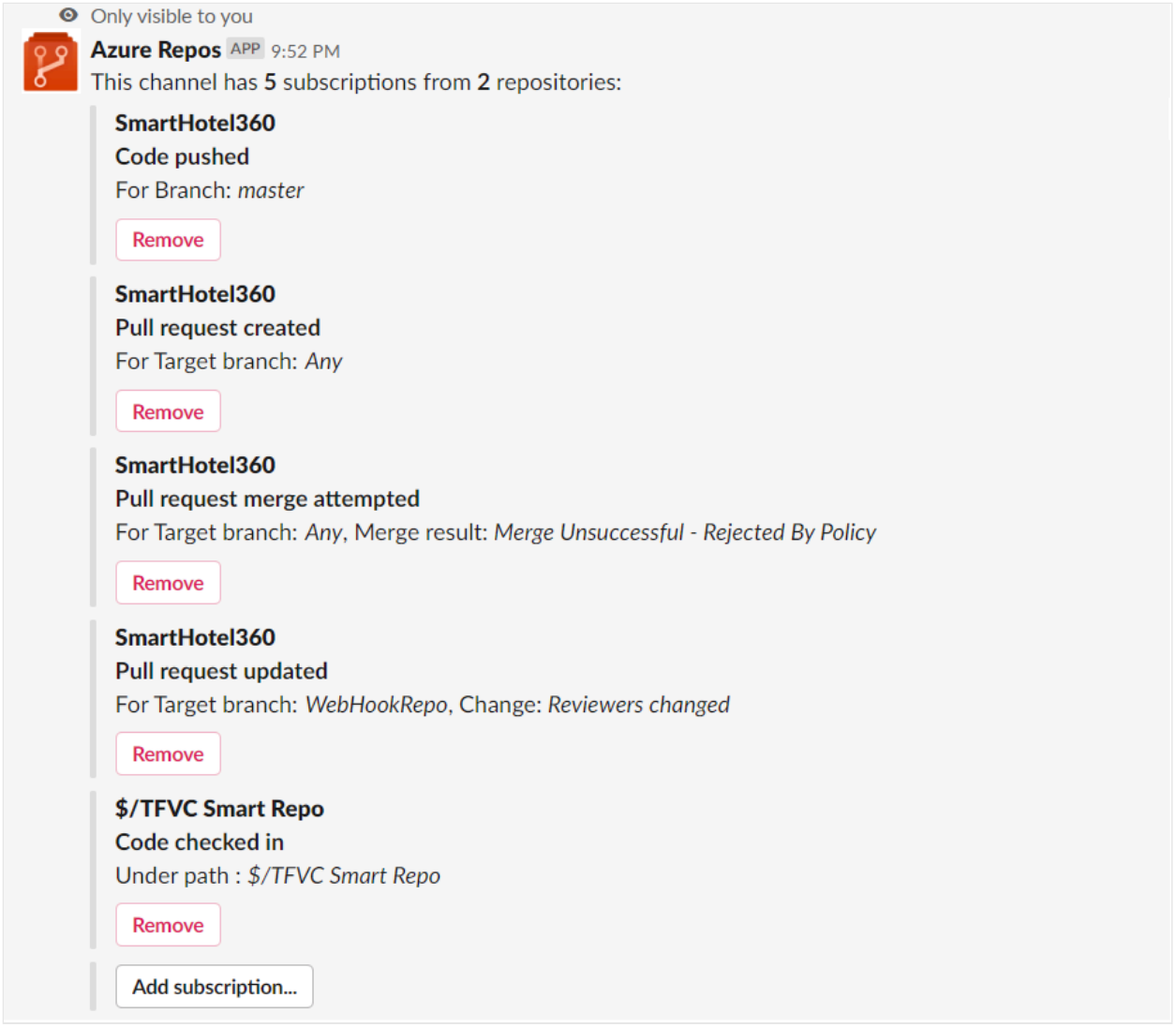
slash

```
/azrepos subscriptions
```

This command lists all the current subscriptions for the channel and allows you to add new subscriptions or remove existing ones. When adding subscriptions, you can

customize the notifications you get by using various filters, as described in the following section.

> ⓘ Note
>
> Team administrators can't remove or modify subscriptions created by Project administrators.



Use filters to customize subscriptions

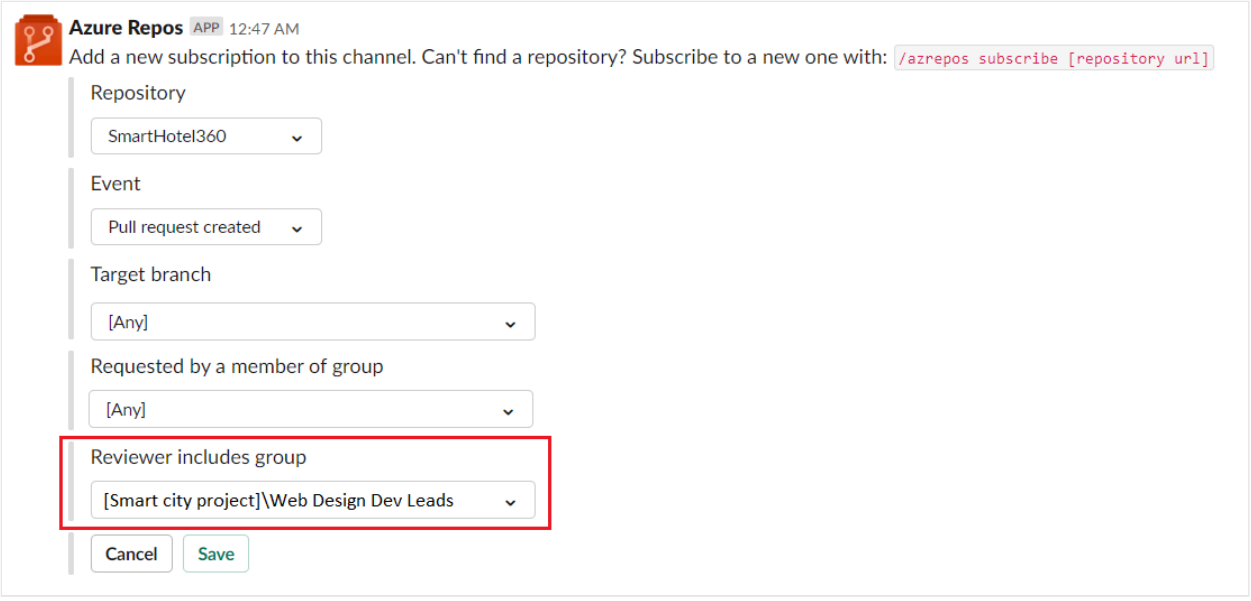
When a user subscribes to a repository using the `/azrepos subscribe` command, a default subscription gets created. Often, users need to customize these subscriptions. For example, users might want to get notified only when PRs have a specific reviewer.

The following steps demonstrate how to customize subscriptions.

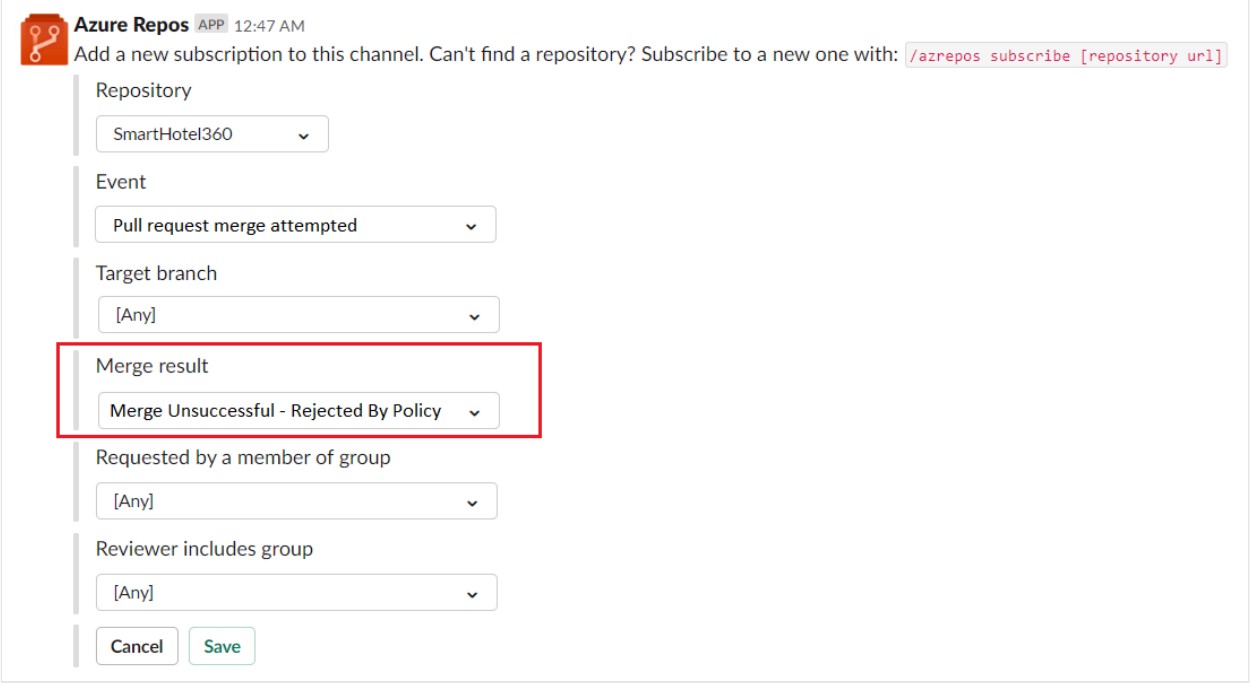
1. Run the `/azrepos subscriptions` command.

2. In the list of subscriptions, if there's a subscription that is unwanted or must be modified (Example: creating noise in the channel), select the **Remove** button.
3. Select the **Add subscription** button.
4. Select the required repository and the desired event.
5. Select the appropriate filters.

Example: Get notifications only when my team is in the reviewer list for a PR



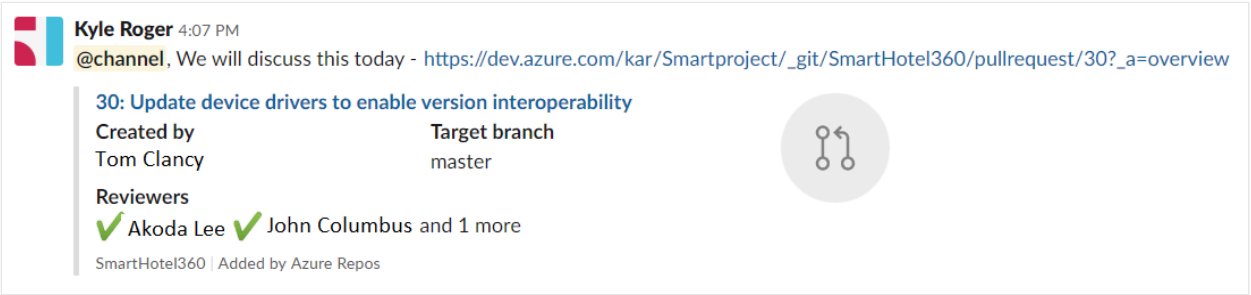
Example: Tell me when merge attempts fail due to a policy violation



> **Note**
>
> - All the filters are typically drop-downs. But if the drop-down has greater than 100 items, then users must enter the values manually.
> - For the TFVC **Code Checked in** event, the filter **Under path** must be of the format `$/myproject/path`.

Preview pull request URLs

When a user pastes the URL of a PR, a preview display like the following image, which helps to keep PR-related conversations contextual and accurate.



Once users sign in, this feature works for all channels in a workspace.

Remove subscriptions and repositories from a channel

Use the following command to clean up your channel by removing repositories and subscriptions.

slash

```
/azrepos unsubscribe all [project url]
```

For example, the following command deletes all the subscriptions related to any repository in the project and removes the repositories from the channel. Only project admins can run this command.

slash

```
/azrepos unsubscribe all https://dev.azure.com/myorg/myproject
```

Command reference

The following table lists all the `/azrepos commands` you can use in your Slack channel.

Expand table

Slash command	Functionality
/azrepos subscribe [repository url/ project url]	Subscribe to a repository or all repositories in a project for notifications
/azrepos subscriptions	Add or remove subscriptions for this channel
/azrepos signin	Sign in to your Azure Repos organization
/azrepos signout	Sign out from your Azure Repos organization
/azrepos feedback	Report a problem or suggest a feature
/azrepos unsubscribe all [project url]	Remove all repositories (belonging to a project) and their associated subscriptions from a channel

Notifications in private channels

The Azure Repos app can help you monitor the repository events in your private channels, too. Invite the bot to your private channel by using `/invite @azrepos`. Then, you can manage your notifications the same way you would for a public channel.

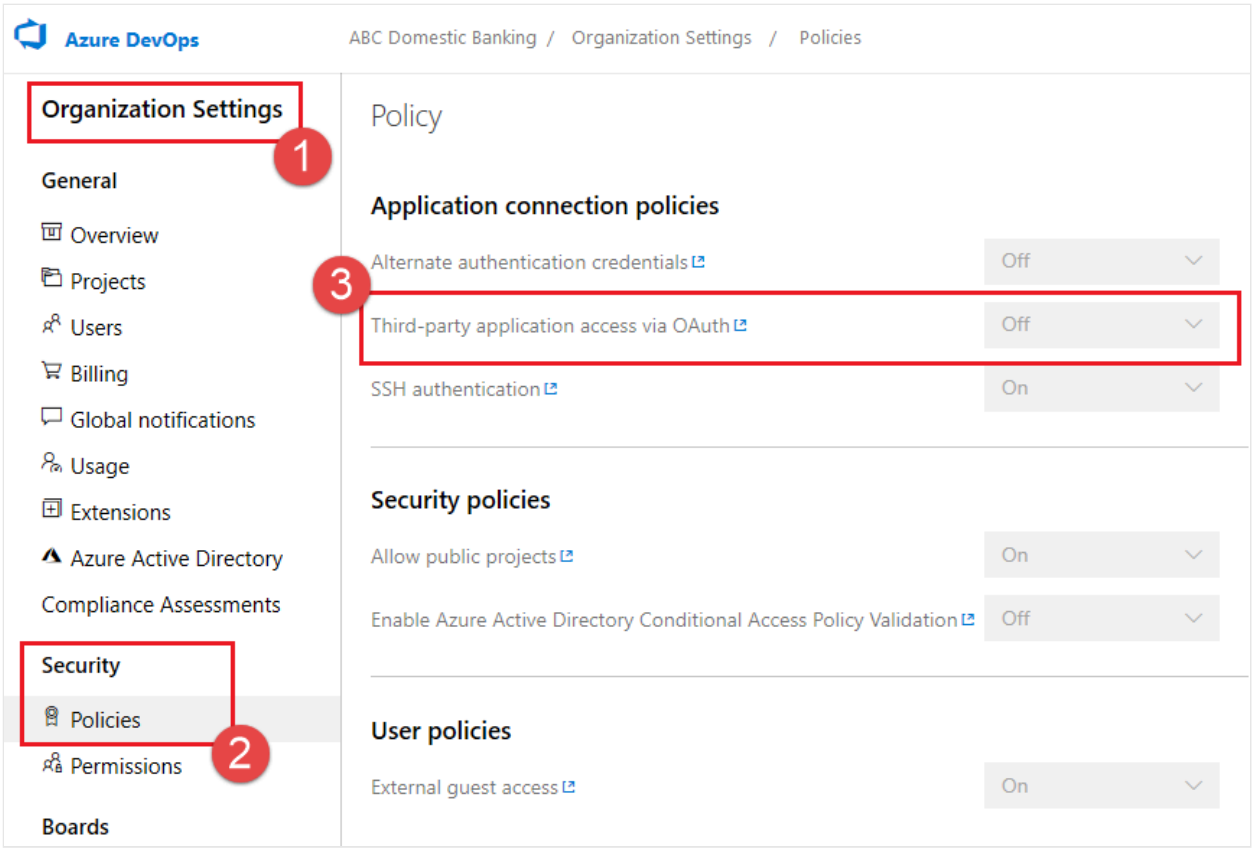
Troubleshoot

If you're experiencing the following errors when using the Azure Repos App for Slack, follow the procedures in this section.

- Sorry, something went wrong. Please try again.
- Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

Sorry, something went wrong. Please try again.

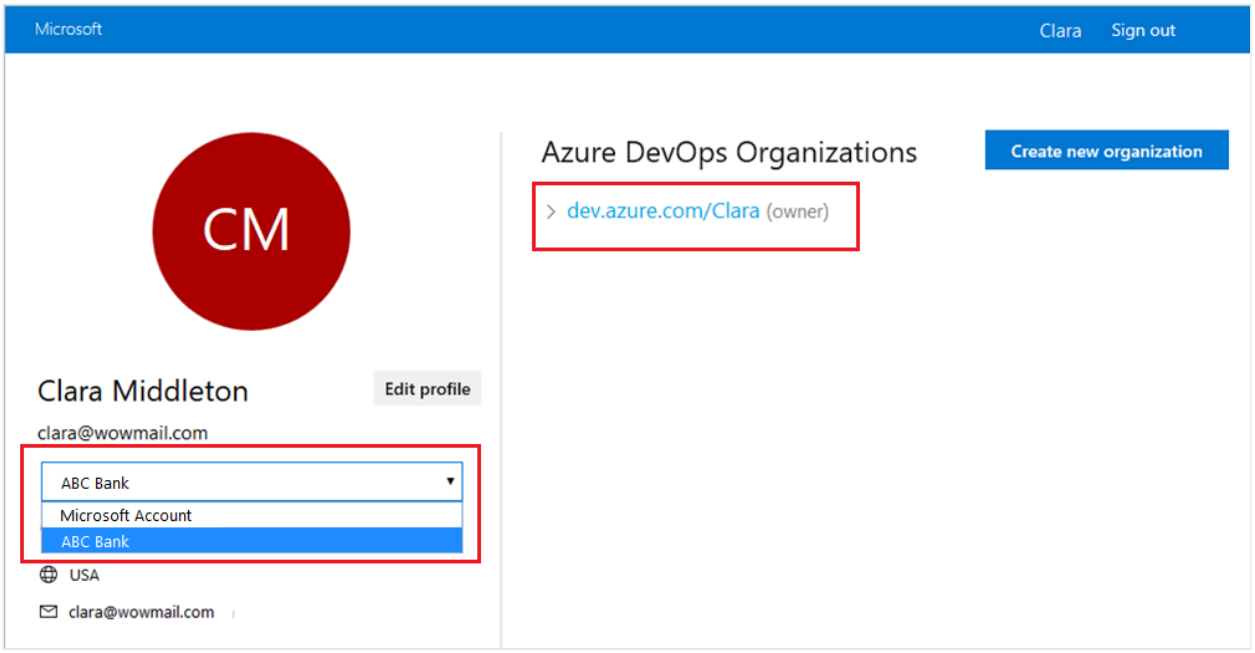
The Azure Repos app uses the OAuth authentication protocol, and requires Third-party application access via OAuth for the organization to be enabled. To enable this setting, navigate to **Organization Settings** > **Security** > **Policies**, and set the **Third-party application access via OAuth for the organization** setting to **On**.



Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

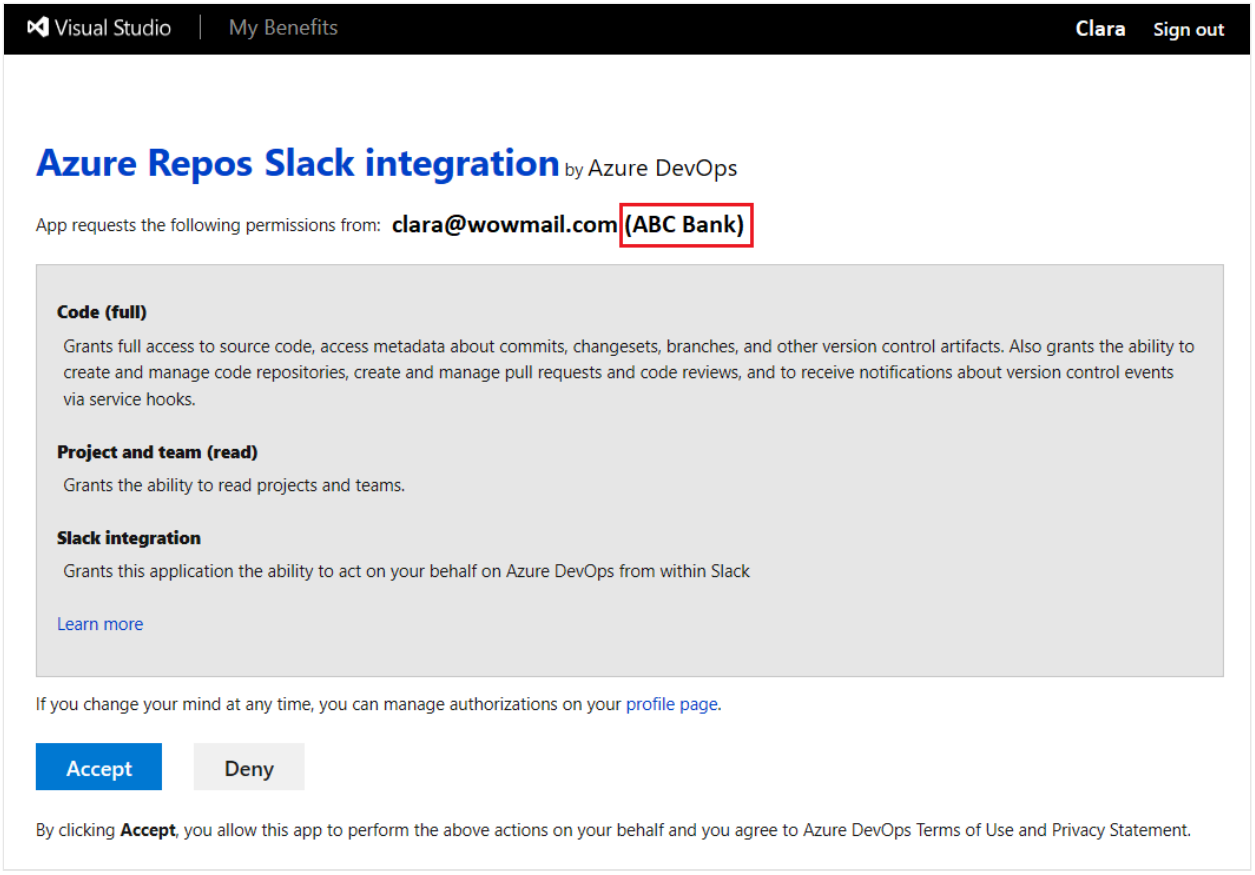
Sign out of Azure DevOps by navigating to `https://aka.ms/VsSignout` using your browser.

Open an **In private** or **incognito** browser window and navigate to `https://aex.dev.azure.com/me` and sign in. In the dropdown under the profile icon to the left, select the directory that contains the organization containing the repository to which you wish to subscribe.



In the **same browser**, start a new tab, navigate to `https://slack.com`, and sign in to your work space (**use web client**). Run the `/azrepos signout` command followed by the `/azrepos signin` command.

Select the `Sign in` button and you're redirected to a consent page like the one in the following example. Ensure that the directory shown beside the email is same as what was chosen in the previous step. Accept and complete the sign in process.



If these steps don't resolve your authentication issue, reach out to us at Developer Community.

Related articles

- Integrate Azure Boards with Slack
- Integrate Azure Pipelines with Slack
- Create a service hook with Slack

Feedback

Was this page helpful? Yes No

Provide product feedback

Use Azure Repos with Microsoft Teams

Article • 05/22/2023

Azure DevOps Services

If you use Microsoft Teams and Azure Repos, you can use the Azure Repos app for Teams to monitor your repos. The app supports monitoring both Git and Team Foundation Version Control (TFVC) repos, but it doesn't support integration with GitHub repos.

In this article, learn how to do the following tasks:

- ✓ Add the Azure Repos app to your team in Microsoft Teams
- ✓ Connect the Azure Repos app to your repos
- ✓ Manage subscriptions to repo related events in your channel
- ✓ Search and share PR info using compose extension
- ✓ Preview PR URLs
- ✓ Remove subscriptions and repos from a channel

Prerequisites

- Manage your subscription, so you receive notifications in your channel whenever code gets pushed or checked in, or when a pull request (PR) gets created, updated, or merged. To create subscriptions for repo-related events, you must be a member of the **Project Administrators** group, or a team administrator. To get added, see Change project-level permissions or Add a team administrator.
- To receive notifications, enable the **Third-party application access via OAuth** setting for the Azure DevOps organization. See Change application access policies for your organization.

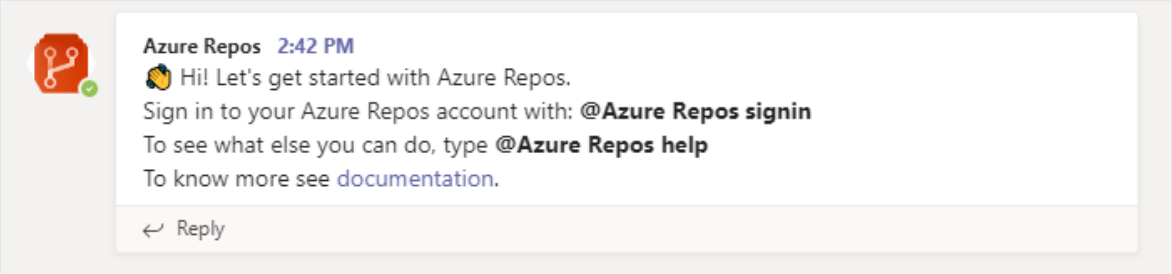
> **Note**
>
> - Notifications are currently not supported inside direct messages.
> - You can only link the Azure Repos app for Microsoft Teams to a project hosted on Azure DevOps Services.

Add the Azure Repos app to a team in Microsoft Teams

1. Go to the Azure Repos app in Microsoft Teams, Microsoft Teams > Azure Repos.

2. Select **Add** or if you already downloaded the app, select the dropdown menu next to **Open**, and then select **Add to a team**.

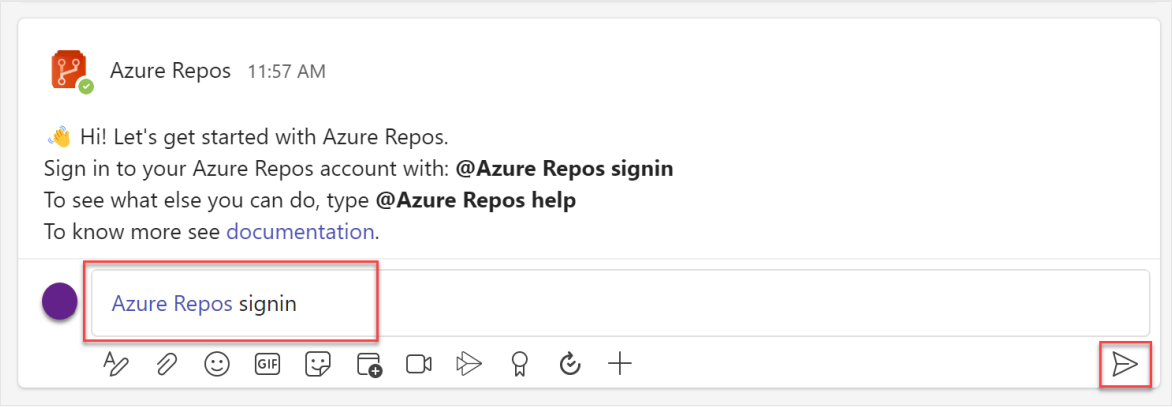
3. Enter a team or channel name, and then select **Set up a bot**.

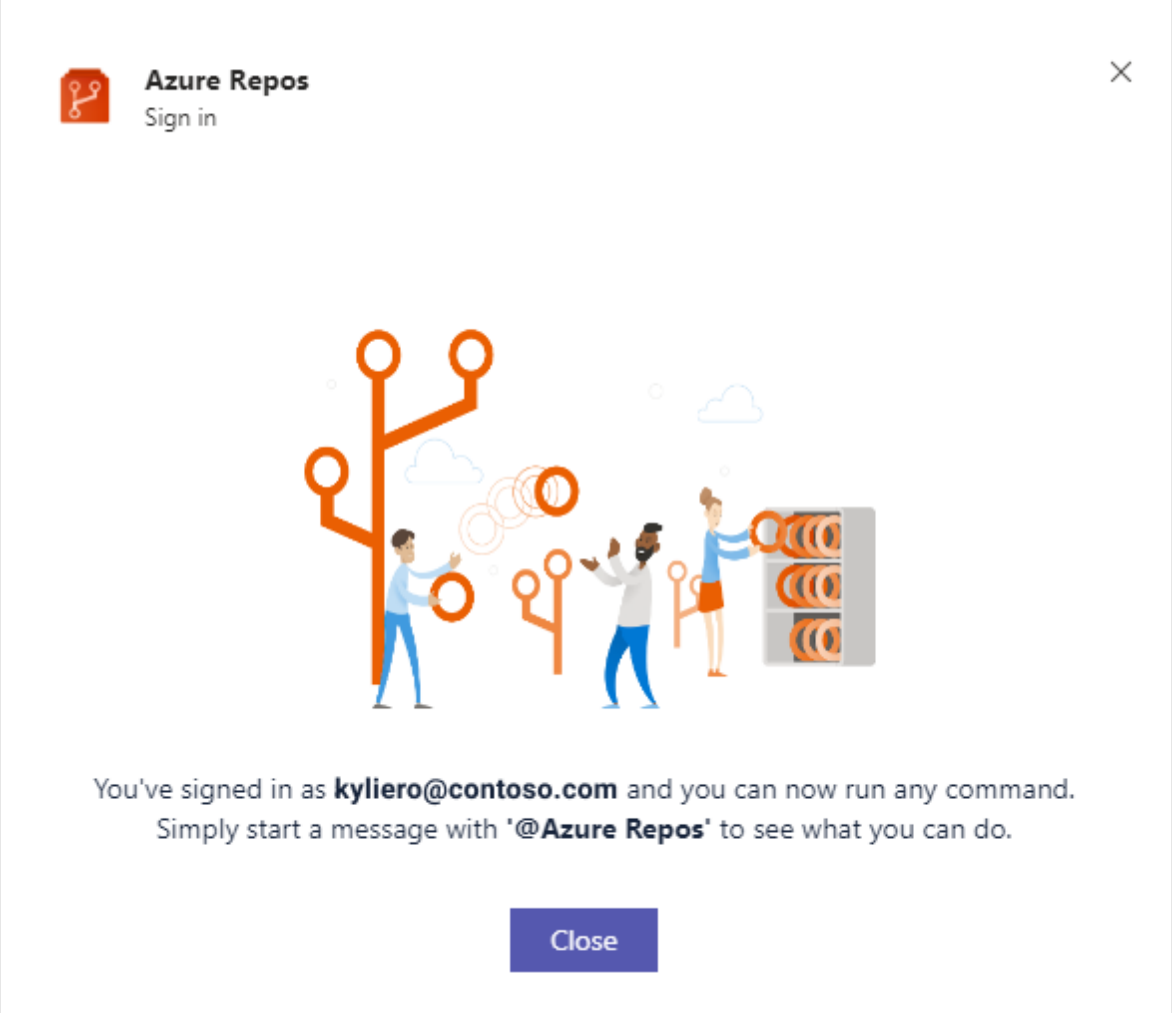


Connect the Azure Repos app to your repos

1. Once the app is installed in your team, enter the following text into the reply field: `@azure repos signin`

   If your Microsoft Teams and Azure Boards are in different tenants, select **Sign in with different email**.



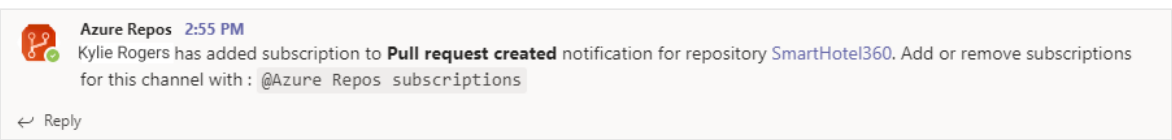


2. To monitor all Git repos in a project, enter `@azure repos subscribe [project url]` into the channel. Be sure to add your project URL. The project URL can be to any page within your project (except URLs to repos).

   You can also monitor a specific repo using: `@azure repos subscribe [repo url]`.

   The repo URL can be to any page within your repo that has your repo name, for example, `@azure repos subscribe https://dev.azure.com/myorg/myproject/_git/myrepo`, or for TFVC repos: `@azure repos subscribe https://dev.azure.com/myorg/myproject/_versionControl`.

3. The `subscribe` command gets you started with a default subscription. For Git repos, the channel gets subscribed to the **Pull request created** event (with target branch = main). For TFVC repos, the channel is subscribed to the **Code checked in** event.

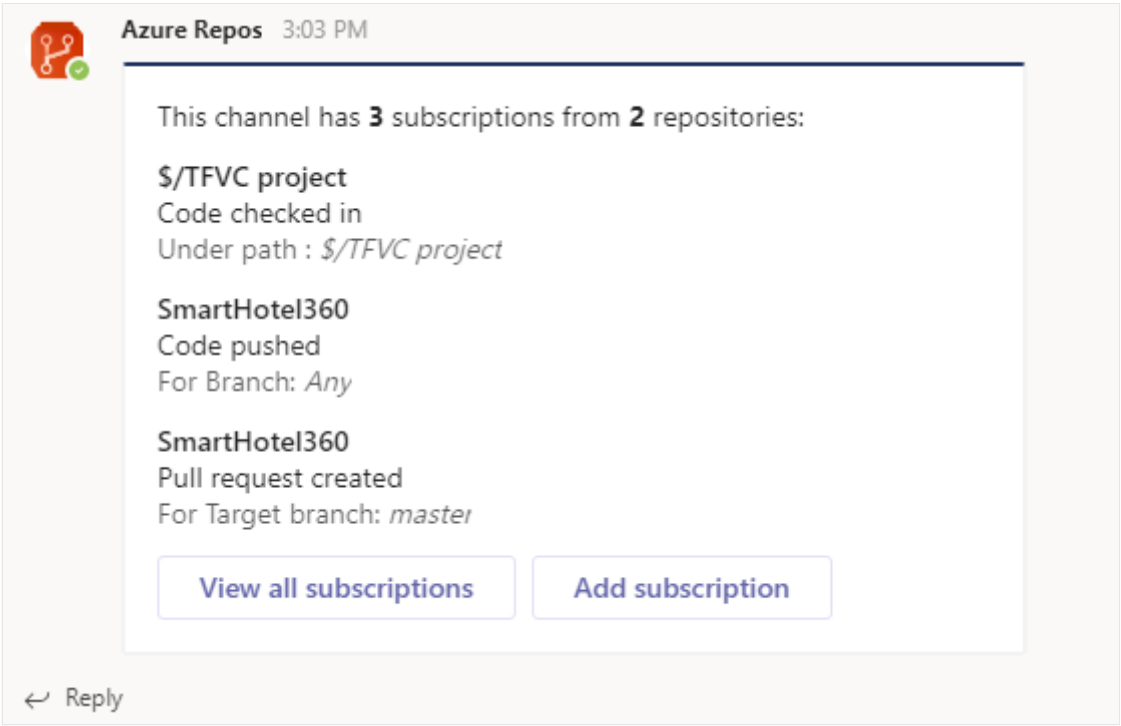


Manage subscriptions

To view, add, and remove subscriptions for a channel, enter the following text: `@azure repos subscriptions`.

You see a list of all the current subscriptions for the channel and you can add new subscriptions or remove existing ones. Customize your notifications with various filters, as described in the following section.

> **Note**
>
> Team administrators can't remove or modify subscriptions created by Project administrators.



Use filters to get only notifications that you want

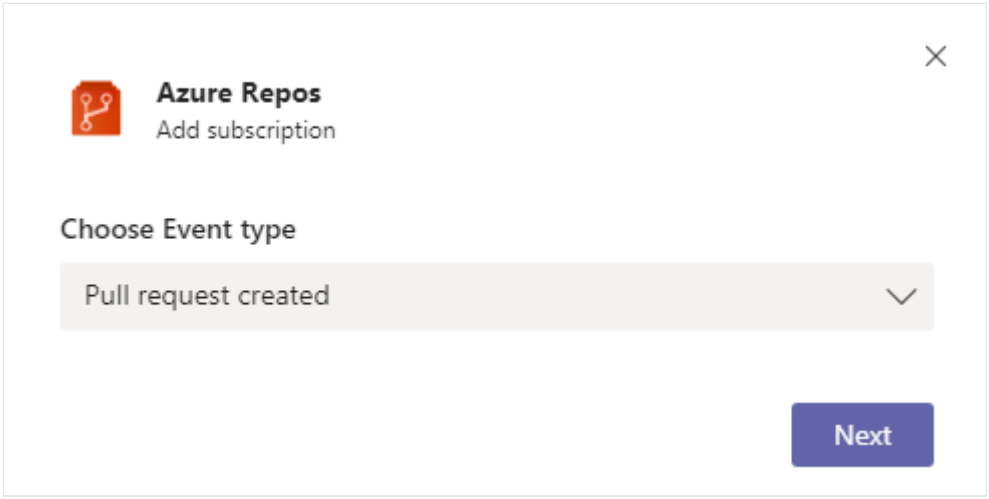
When you subscribe to a repo with `@azure repos subscribe`, a default subscription gets created with no filters applied. Often, users need to customize these subscriptions to be notified only when certain conditions are met. The following screenshots show an example of setting up notifications only when our team is in the reviewer list for a PR.

1. Enter the following text into your channel: `@azure repos subscriptions`.

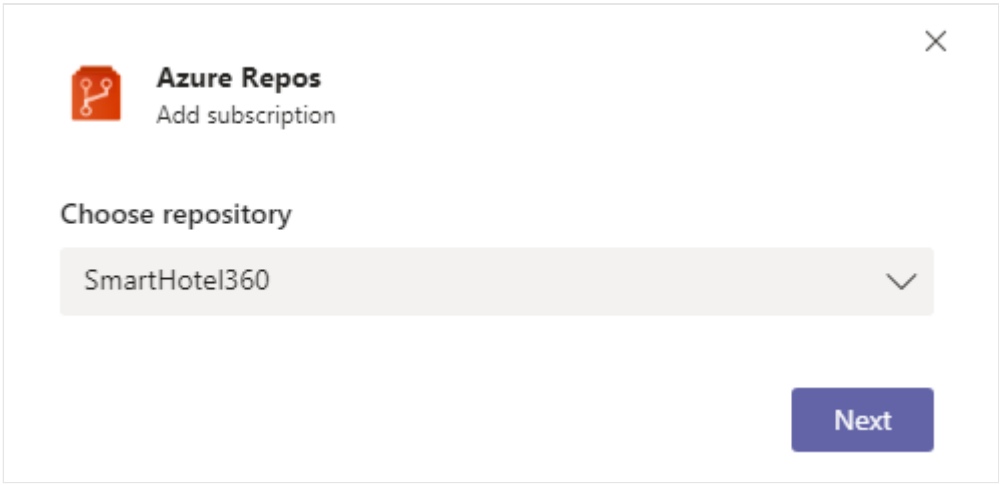
2. In the list of subscriptions, if there's a subscription that you don't want or must be modified, select **Remove** to delete it.

3. Select the **Add subscription** button.

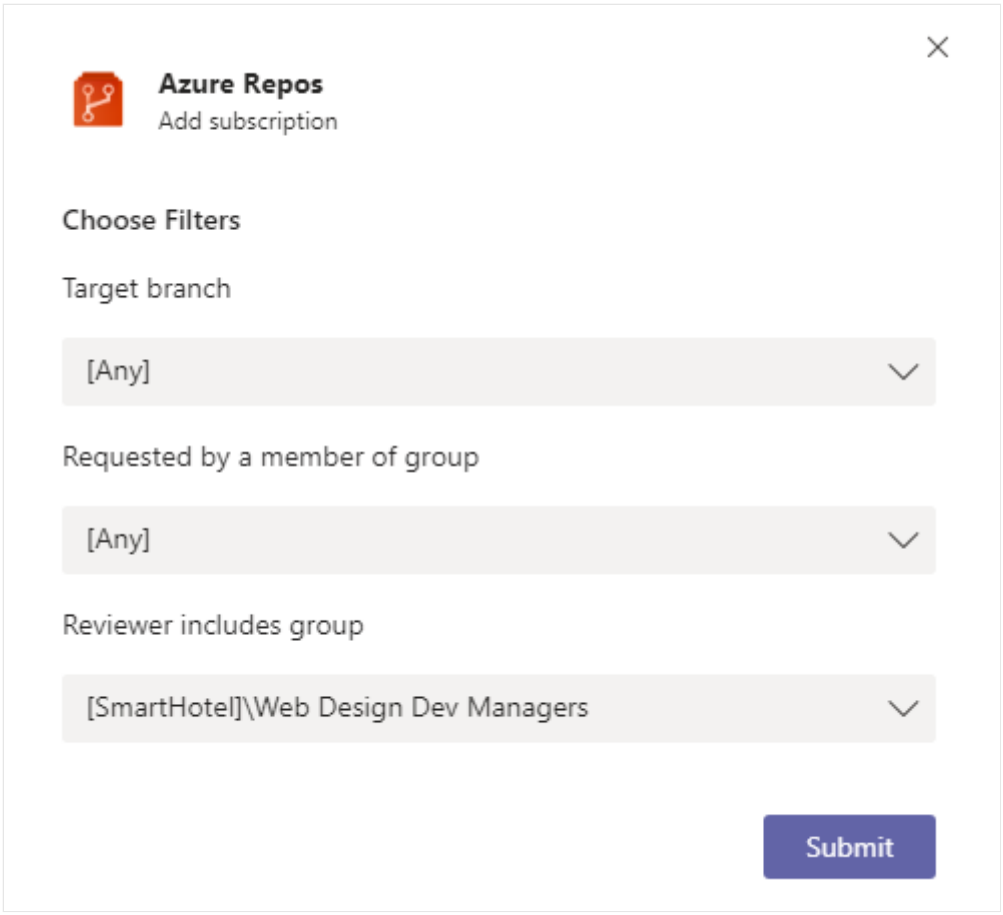
4. Choose an event type, and then select **Next**.



5. Choose a repo, and then select **Next**.



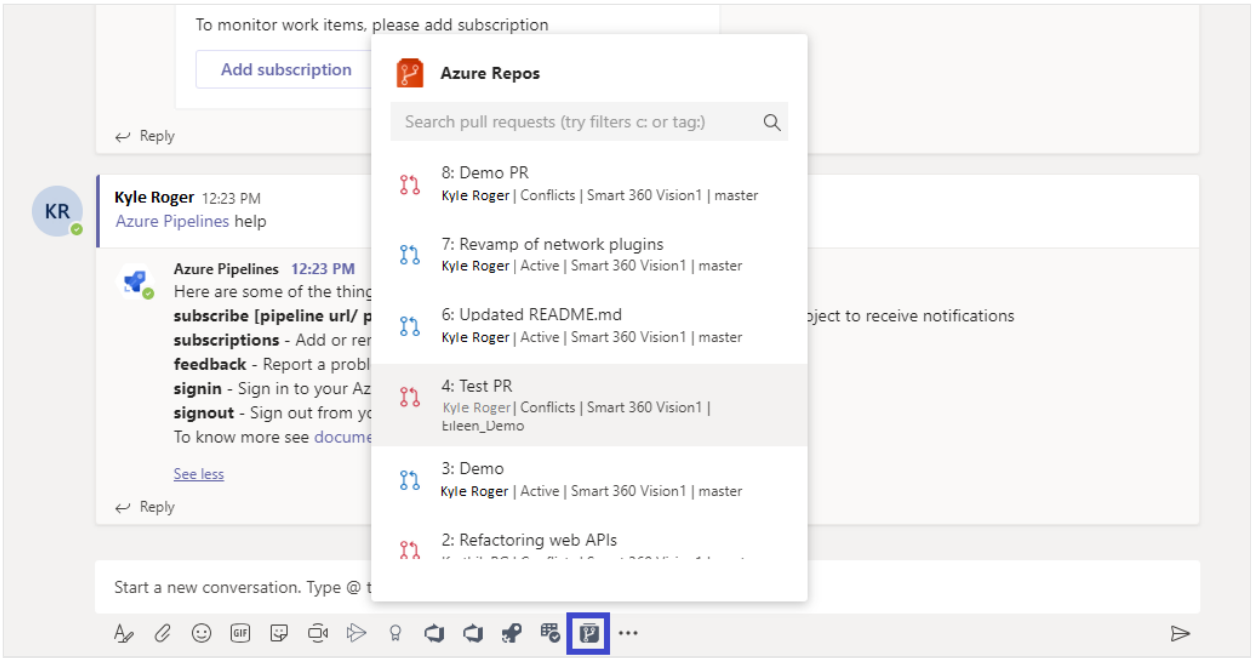
6. Select the appropriate filters to customize your subscription, and then select **Submit**.



> **Note**
>
> - All the filters are typically drop-downs. However, if the drop-down has greater than 100 items, you must manually enter the values.
> - For the TFVC **Code Checked in** event, the filter **Under path** must be of the format `$/myproject/path`.

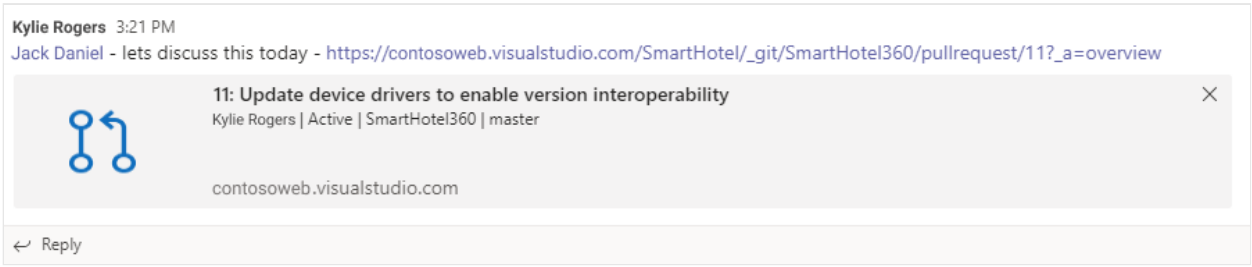
Search and share pull request information using compose extension

To help users search and share information about pull requests, Azure Repos app for Microsoft Teams supports a compose extension. You can search for pull requests by ID or name. For the extension to work, sign in to the Azure Repos project by entering `@azure repos signin` or by signing into the compose extension directly.



Preview pull request URLs

When you paste the URL of a PR, a preview shows like the one in the following image, which helps to keep PR-related conversations contextual and accurate. You must be signed in, and then you can preview PRs for URLs in all channels in a Team.



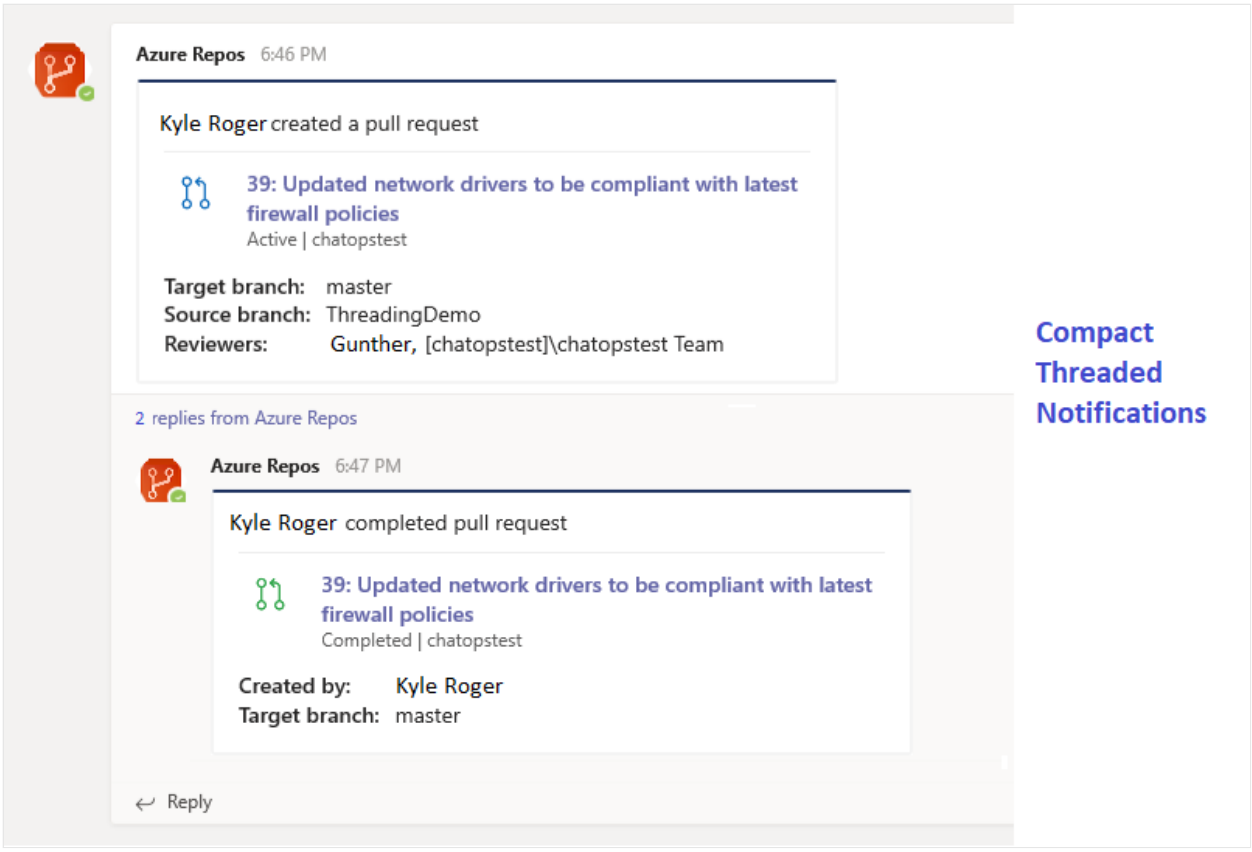
Remove subscriptions and repos from a channel

To delete all the subscriptions related to any repo in the project and remove the repos from the channel, enter the following text into Teams: `@azure repos unsubscribe all [project url]`. Make sure to enter the project URL. Only project admins can do this task.

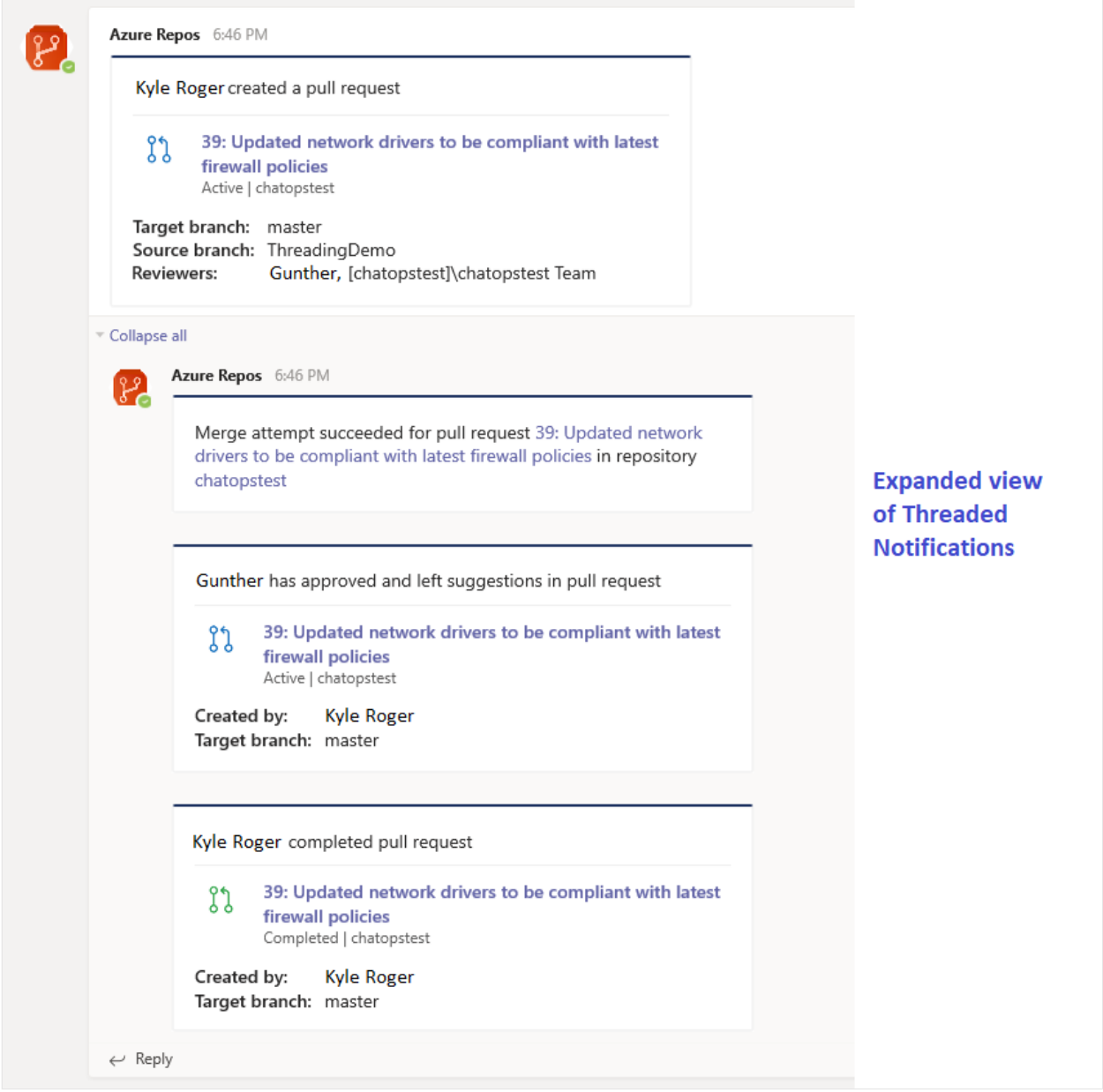
Threaded notifications

To link a set of related notifications and also to reduce the space occupied by notifications in a channel, notifications get threaded. All notifications linked to a particular pull request are linked together.

Compact view of threaded notifications



Expanded view of threaded notifications



Command reference

The following table lists all the `azure repos` commands you can use in your Teams channel.

Command	Functionality
`@azure repos subscribe [repo url/ project url]`	Subscribe to a repo or all repos in a project to receive notifications
`@azure repos subscriptions`	Add or remove subscriptions for this channel
`@azure repos signin`	Sign in to your Azure Repos organization
`@azure repos signout`	Sign out from your Azure Repos organization

Command	Functionality
`@azure repos feedback`	Report a problem or suggest a feature
`@azure repos unsubscribe all [project url]`	Remove all repos (belonging to a project) and their associated subscriptions from a channel

Multi-tenant support

If you're using a different email or tenant for Microsoft Teams and Azure DevOps, do the following steps to sign in, based on your use case.

Use case	Email ID + Microsoft Teams tenant	Email ID + Azure DevOps tenant	Steps
1	email1@abc.com (tenant 1)	email1@abc.com (tenant 1)	Select **Sign in**.
2	email1@abc.com (tenant 1)	email1@abc.com (tenant 2)	Sign in to Azure DevOps. In the same browser, start a new tab and go to https://teams.microsoft.com/. Run the signin command and select **Sign in**.
3	email1@abc.com (tenant 1)	email2@pqr.com (tenant 2)	Select **Sign in with different email address**, and then in the email ID picker use the email2 to sign in to Azure DevOps.
4	email1@abc.com (tenant 1)	email2@pqr.com (non default tenant 3)	This scenario isn't supported.

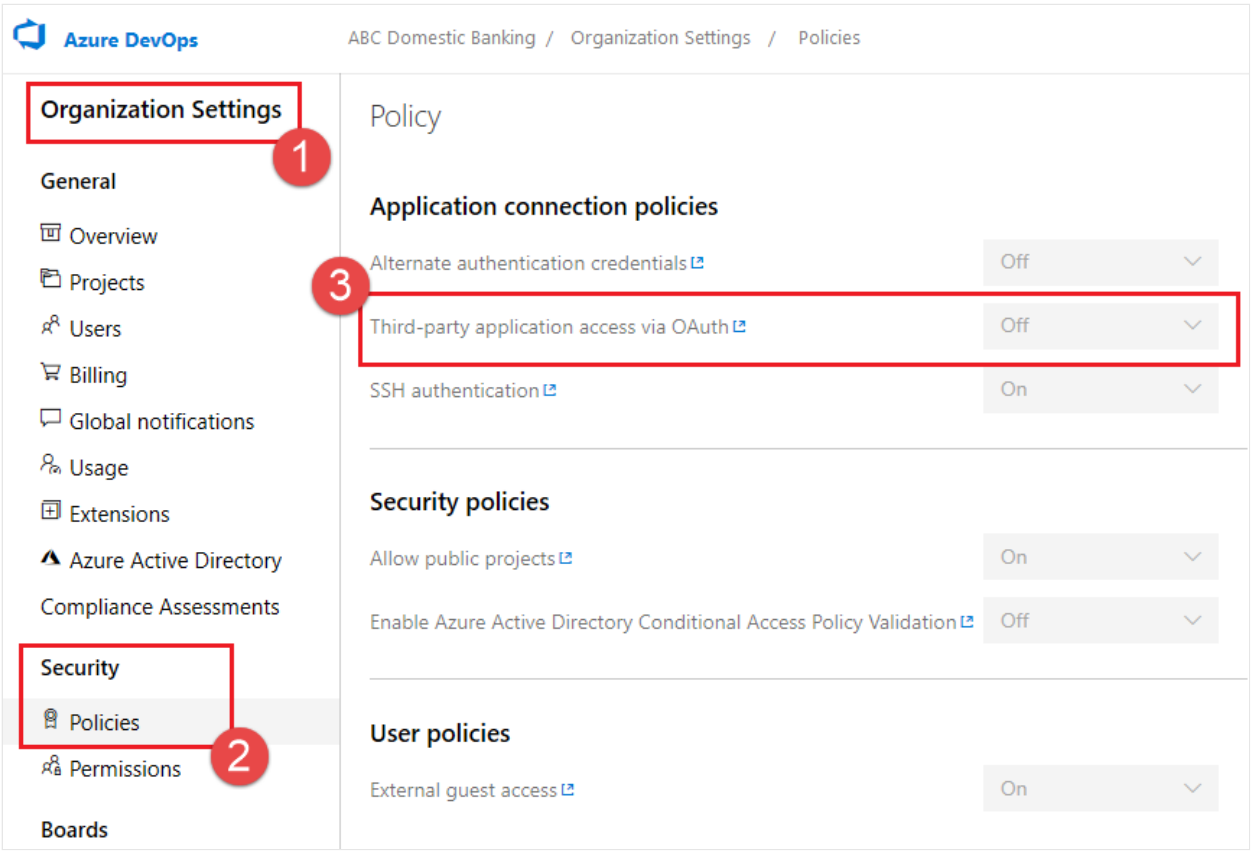
Troubleshoot

If you're experiencing the following errors when using the Azure Repos App, follow the procedures in this section.

- Sorry, something went wrong. Please try again.
- Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

Sorry, something went wrong. Please try again.

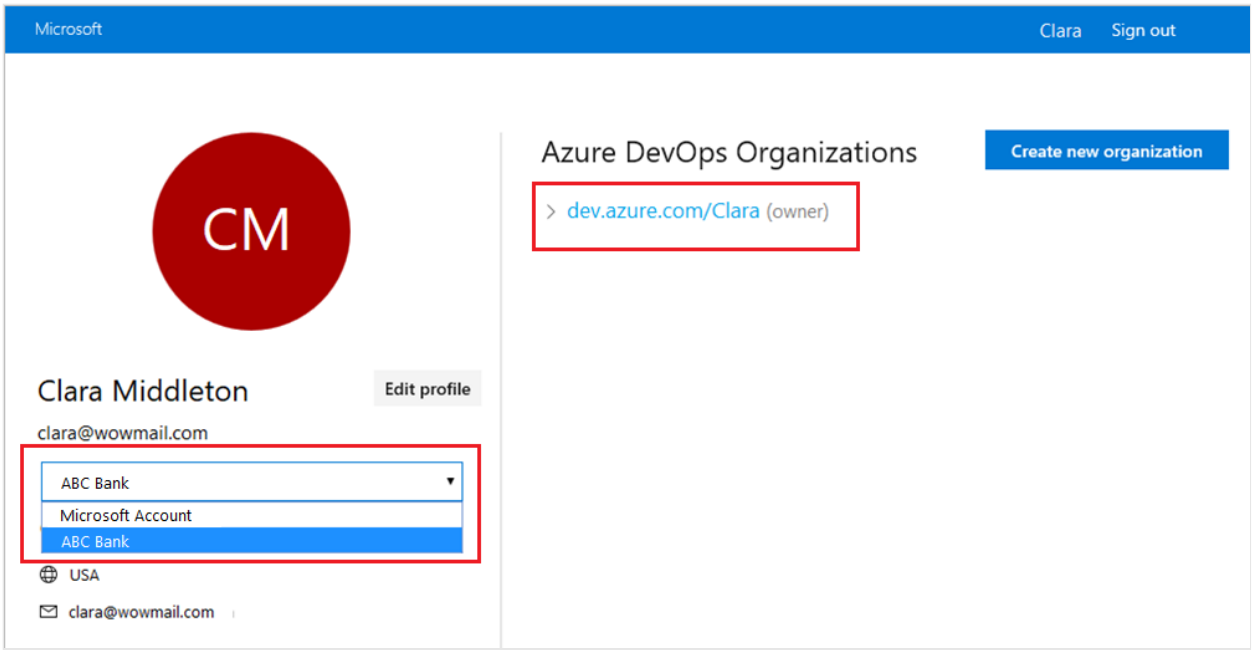
The Azure Repos app uses the OAuth authentication protocol, and requires Third-party application access via OAuth for the organization to be enabled. To enable this setting, navigate to **Organization Settings** > **Security** > **Policies**, and set the **Third-party application access via OAuth for the organization** setting to **On**.



Configuration failed. Please make sure that the organization '{organization name}' exists and that you have sufficient permissions.

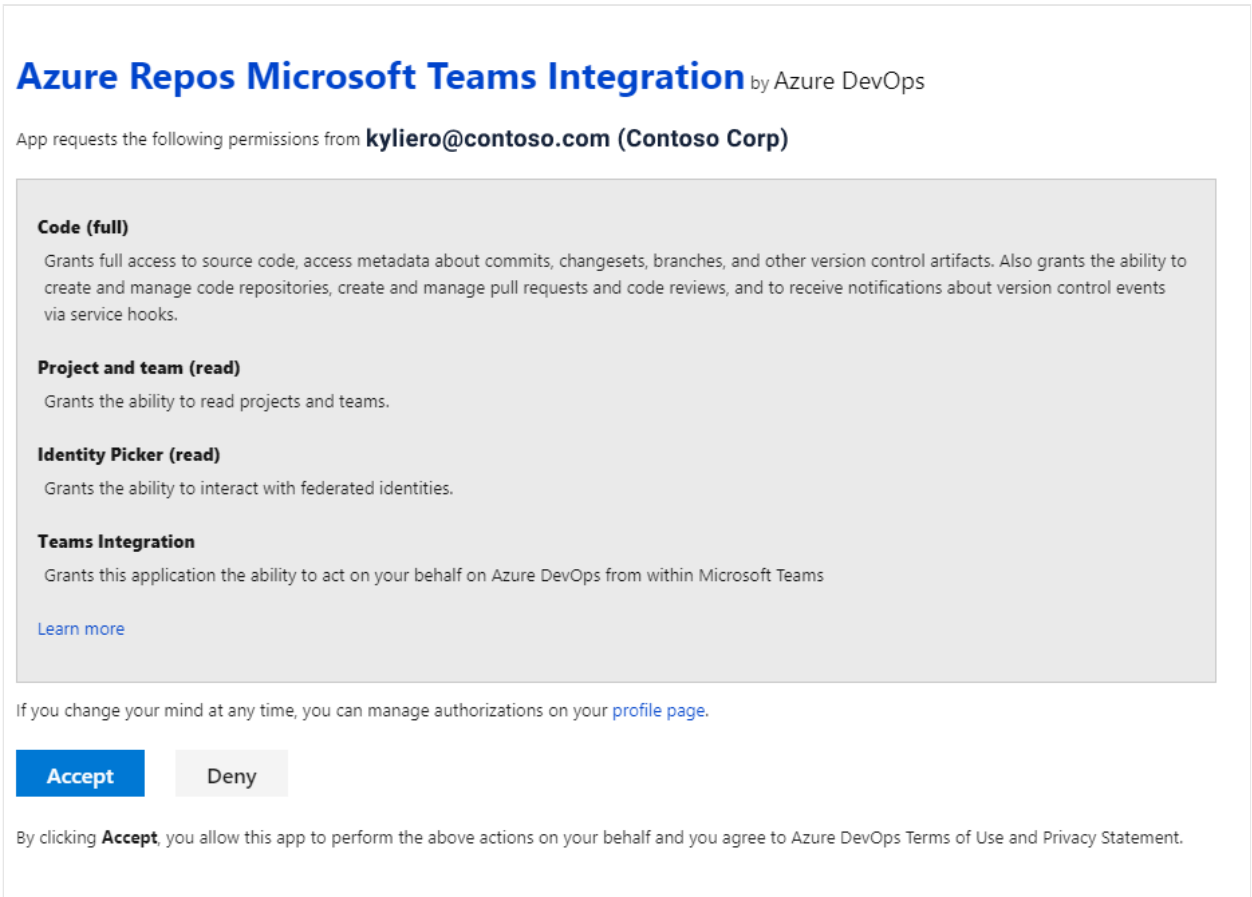
Sign out of Azure DevOps by navigating to `https://aka.ms/VsSignout` using your browser.

Open an **In private** or **incognito** browser window and navigate to `https://aex.dev.azure.com/me` and sign in. In the dropdown under the profile icon to the left, select the directory that contains the organization containing the repository to which you wish to subscribe.



In the **same browser**, start a new tab and sign in to `https://teams.microsoft.com/`. Run the `@Azure Repos signout` command and then run the `@Azure Repos signin` command in the channel where the Azure Repos app for Microsoft Teams is installed.

Select the `Sign in` button and you're redirected to a consent page like the one in the following example. Ensure that the directory shown beside the email is same as what was chosen in the previous step. Accept and complete the sign-in process.



If these steps don't resolve your authentication issue, reach out to us at Developer Community.

Related articles

- Azure Boards with Teams
- Azure Pipelines with Teams

Use Azure Pipelines with Slack

Article • 07/23/2024

Azure DevOps Services

This article shows you how to use the Azure Pipelines app for Slack to monitor your pipeline events. You can establish and manage subscriptions for pipeline events like builds, releases, and pending approvals. Notifications for these events are delivered directly to your Slack channels.

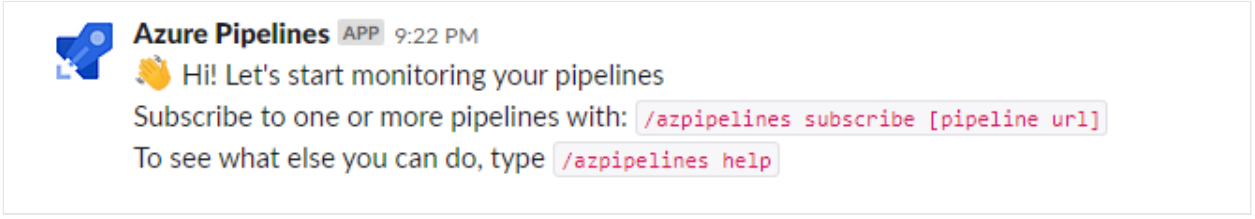
> **Note**
>
> This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. For more information, see Azure DevOps Feature Timeline.

Prerequisites

- A Slack account with permission to install an app to your Slack workspace.
- An Azure DevOps project with **Project Collection Administrators** or **Project Administrators** permissions.

Install the Azure Pipelines app

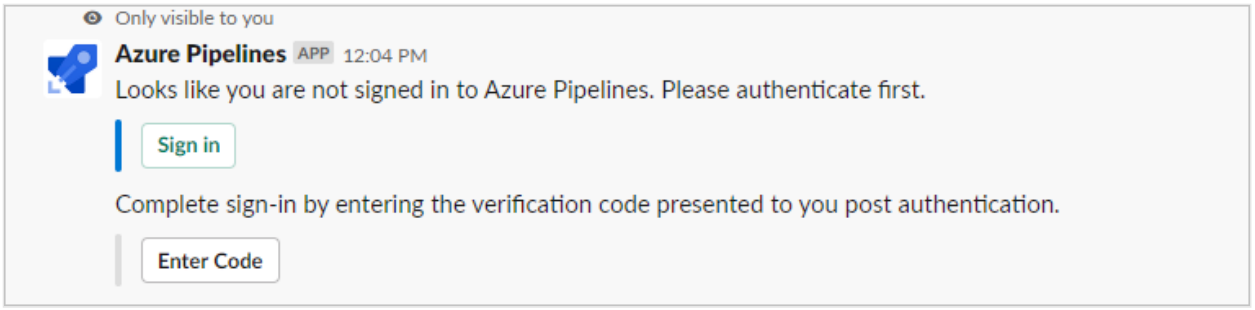
Install the Azure Pipelines Slack app to your Slack workspace. Once the app installs, you see the following welcome message. Enter `/azpipelines` to start interacting with the app.



Connect to your pipeline

Once the app is installed in your Slack workspace, you can connect the app to any pipeline you want to monitor. You must authenticate to Azure Pipelines before running

any commands.



Subscribe to pipelines

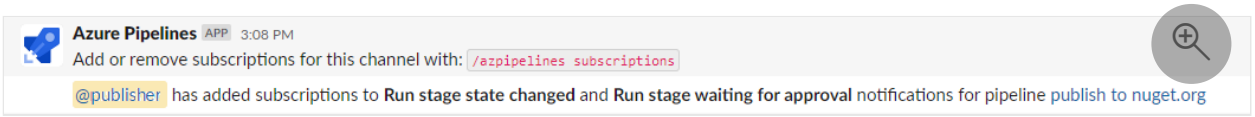
To start monitoring all pipelines in a project, enter `/azpipelines subscribe <project url>` in a channel, replacing `<project url>` with your Azure DevOps project URL. The project URL can link to any page within your project except pipeline pages, for example `/azpipelines subscribe https://dev.azure.com/myorg/myproject/`.

You can monitor a specific pipeline by using `/azpipelines subscribe <pipeline url>`. The pipeline URL can link to any page within your pipeline that has a `definitionId` or a `buildId/releaseId` in the URL. For example:

- `/azpipelines subscribe https://dev.azure.com/myorg/myproject/_build?definitionId=123`
- `/azpipelines subscribe https://dev.azure.com/myorg/myproject/_release?definitionId=123&view=mine&_a=releases`

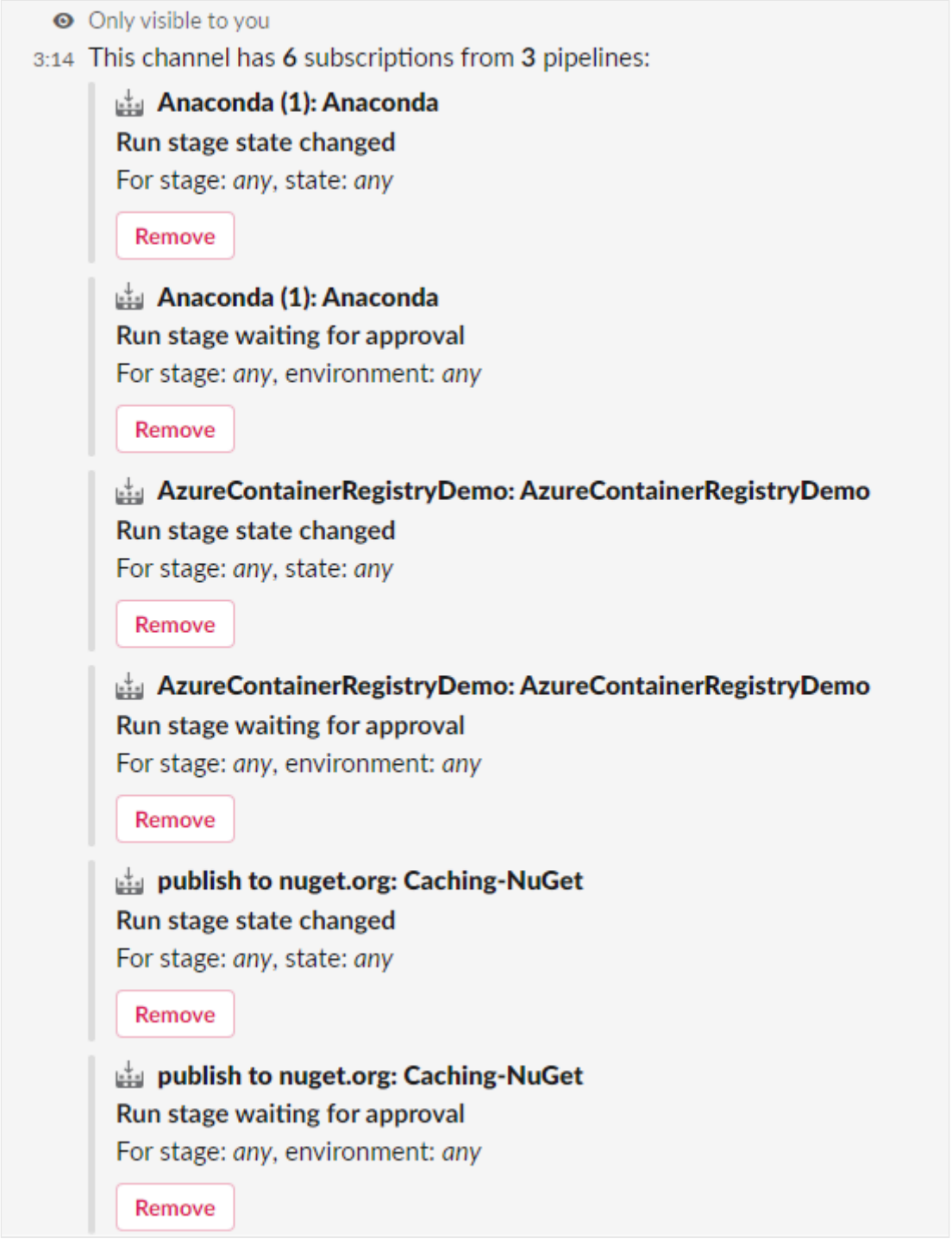
The `subscribe` command subscribes you to the following notifications by default:

- For YAML pipelines:
  - **Run stage state changed**
  - **Run stage waiting for approval**
- For Classic build pipelines, **Builds completed**
- For Classic release pipelines:
  - **Release deployment started**
  - **Release deployment completed**
  - **Release deployment approval pending**



Manage subscriptions

To manage the subscriptions for a channel, enter `/azpipelines subscriptions`. This command lists all the current subscriptions for the channel and lets you add or remove subscriptions.



> ⓘ Note
>
> Team Administrators can't remove or modify subscriptions created by Project Administrators.

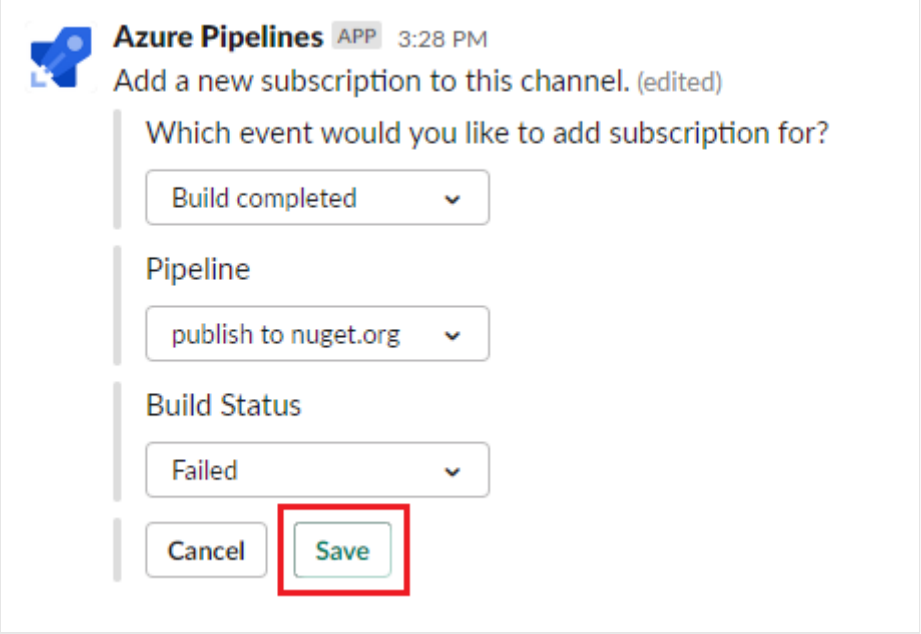
Customize subscriptions

The default subscriptions don't have any filters applied, but you can customize these subscriptions according to your preferences. For instance, you might want to receive notifications only for failed builds or deployments to production. You can apply filters to customize which messages you receive in your channel.

To customize a subscription:

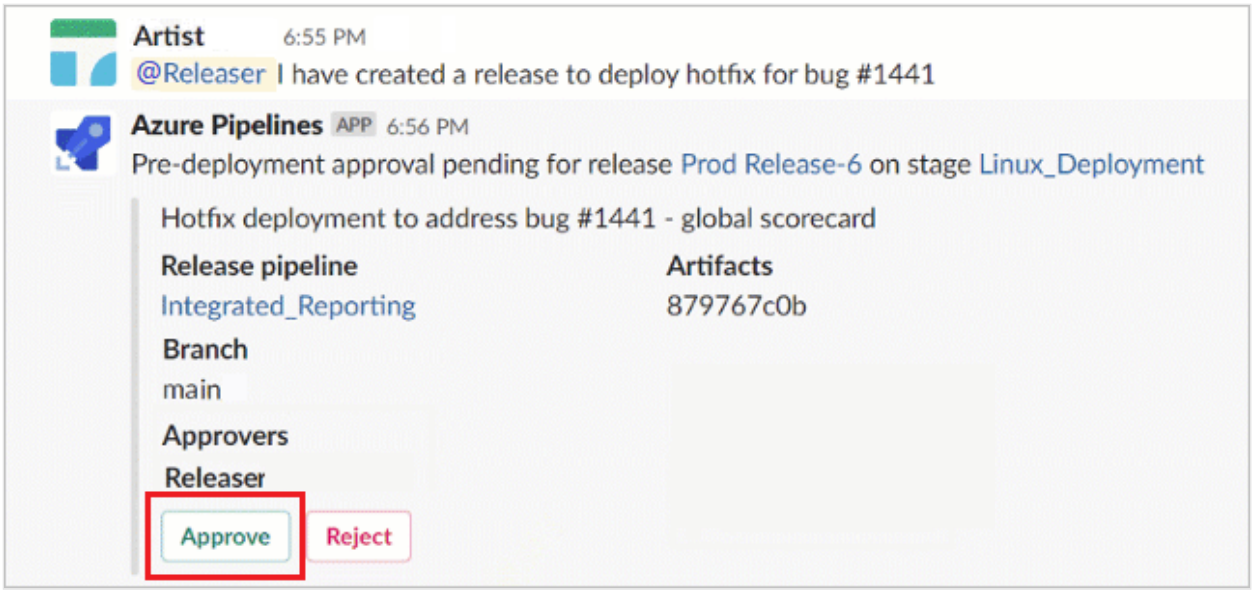
1. Run the `/azpipelines subscriptions` command to list all your subscriptions.
2. Select **Add subscription**.
3. Select the event you want to subscribe to, and then select your desired configuration.
4. Select **Save**.

For example, to get notifications only for failed builds, select **Failed** under **Build status**.

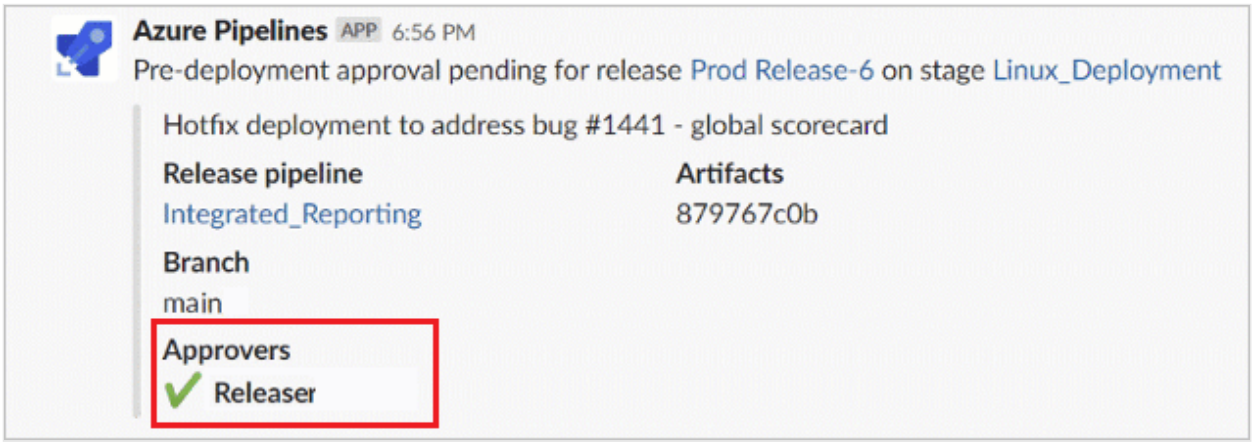


Approve deployments

You can approve deployments from within your Slack channel without going to Azure Pipelines. Subscribe to the **Run stage waiting for approval** notifications for YAML pipelines or the **Release deployment approval pending** notifications for Classic releases. Both of these subscriptions are created by default when you subscribe to a pipeline.



The Azure Pipelines app for Slack lets you handle all the checks and approval scenarios that are available in the Azure Pipelines portal. These scenarios include single approver, multiple approvers, and team-based approval. You can approve requests either individually or on behalf of a team.



Remove all subscriptions

To declutter your channel, you can use the `/azpipelines unsubscribe all <project url>` command to unsubscribe from all pipelines in a project. For example, `/azpipelines unsubscribe all https://dev.azure.com/myorg/myproject`.

> **Important**
>
> Only Project Administrators can run this command.

Command reference

The Azure Pipelines app for Slack supports the following commands:

Expand table

Command	Description
`/azpipelines subscribe <pipeline url or project url>`	Subscribe to a pipeline or all pipelines in a project and receive notifications.
`/azpipelines subscriptions`	Add or remove subscriptions for this channel.
`/azpipelines feedback`	Report a problem or suggest a feature.
`/azpipelines help`	Get help on the commands.
`/azpipelines signin`	Sign in to your Azure Pipelines account.
`/azpipelines signout`	Sign out of your Azure Pipelines account.
`/azpipelines unsubscribe all <project url>`	Remove all project pipelines and their associated subscriptions from a channel.

Notifications in private channels

The Azure Pipelines app can also help you monitor pipelines activity in your private channels. You need to invite the bot to your private channel by using `/invite @azpipelines`. Once you add the bot, you can configure and control your notifications the same way as for a public channel.

Conditions and limitations

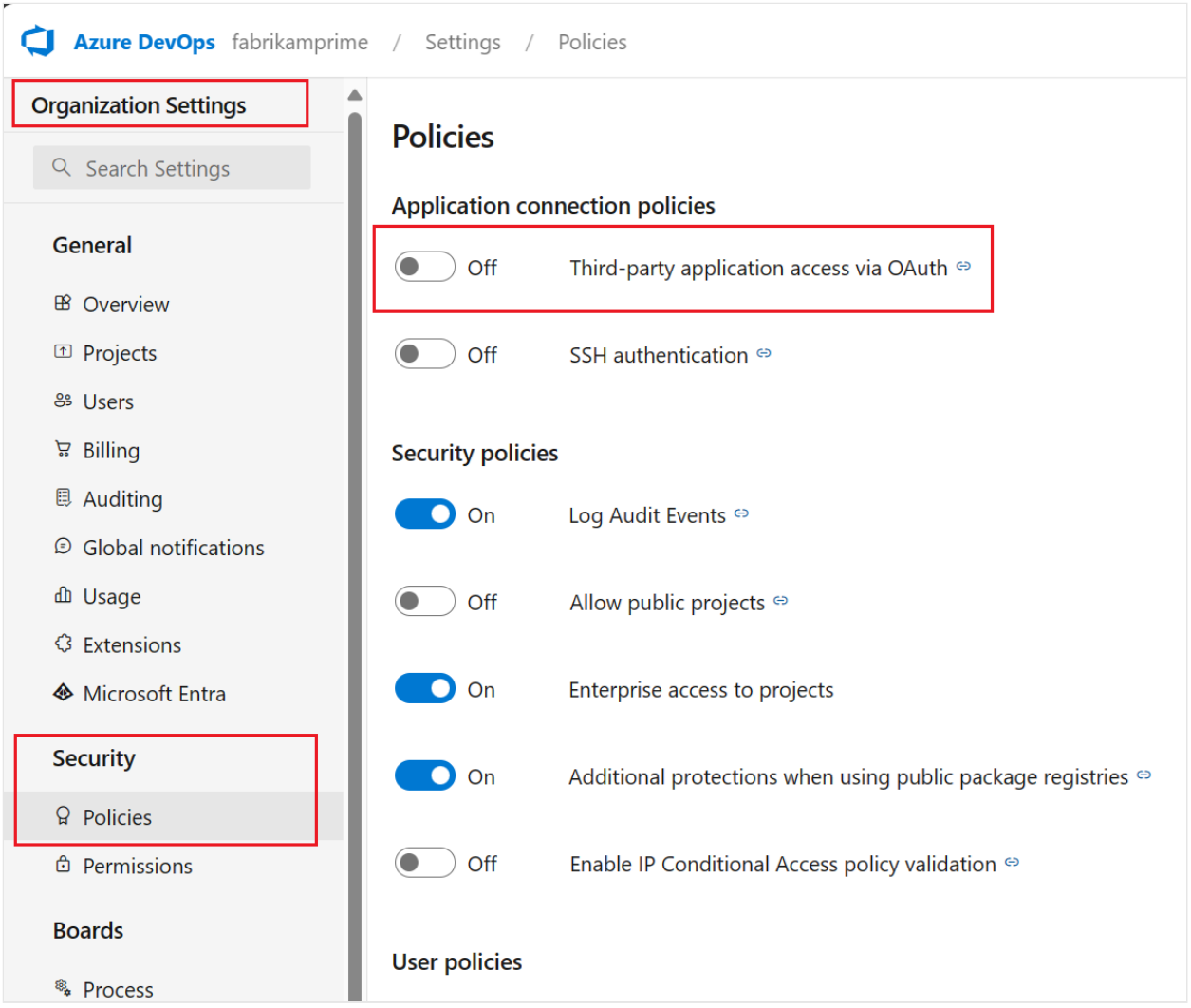
- You can use the Azure Pipelines app for Slack only with Azure DevOps Services.
- To set up the subscriptions, you must be an admin of the project containing the pipeline.
- Notifications aren't supported inside direct messages.
- Deployment approvals that have the **Revalidate identity of approver before completing the approval** policy applied aren't supported.
- To use the app, **Third party application access via OAuth** must be enabled in Azure DevOps **Organization settings** > **Security** > **Policies**.

Troubleshooting

If you get the following errors when using the Azure Pipelines App for Slack, try the procedures in this section.

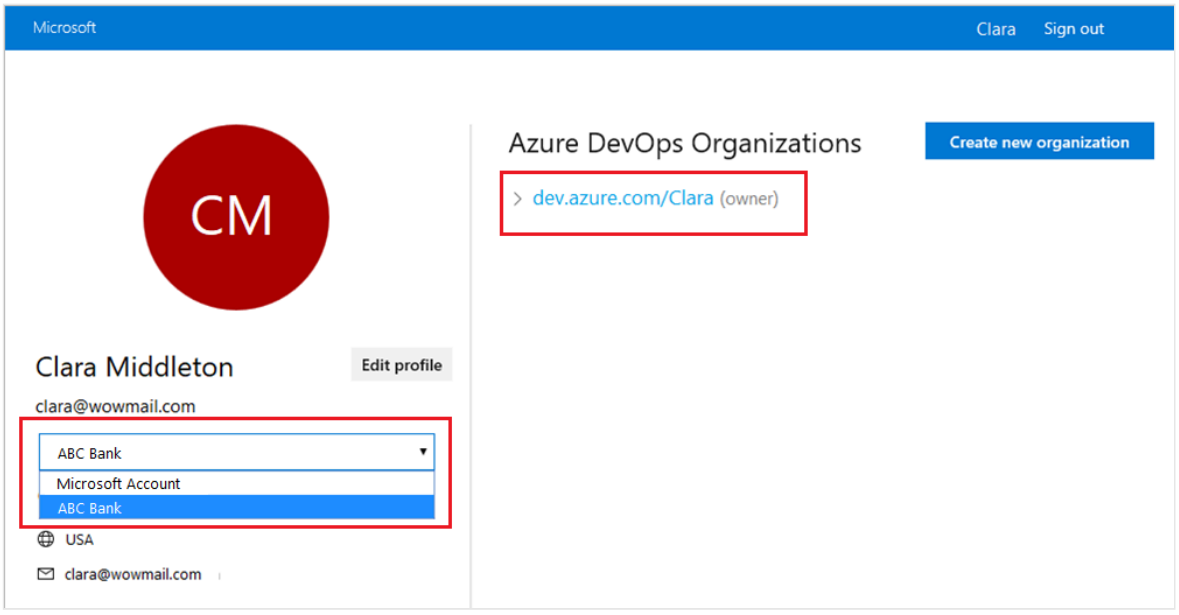
Sorry, something went wrong. Please try again.

The Azure Pipelines app uses the OAuth authentication protocol, and requires Third-party application access via OAuth to be enabled. To enable this setting, navigate to **Organization Settings** > **Security** > **Policies**, and enable **Third-party application access via OAuth**.



Configuration failed. Please make sure that the organization exists and that you have sufficient permissions.

1. Sign out of Azure DevOps by navigating to `https://aka.ms/VsSignout`.
2. In a private/incognito browser window, navigate to `https://aex.dev.azure.com/me` and sign in. Make sure to select the directory containing the organization that has your pipeline.



3. In the same browser, open a new tab and go to `https://slack.com`. Sign in to your workspace using the web client, and then run `/azpipelines signout` followed by `/azpipelines signin`.

4. Select the `Sign in` button. If you're redirected to a consent page, verify that the directory displayed next to your email address matches the one you signed in to.

Related articles

- Azure Boards with Slack
- Azure Repos with Slack
- Service hook for Azure DevOps with Slack

Feedback

Was this page helpful? Yes No

Provide product feedback

Integrate Azure Pipelines with Microsoft Teams

Article • 10/31/2024

> **Note**
>
> Microsoft Teams integration support for service hooks is retiring as of December 31, 2024 for new integrations and January 31, 2025 for existing integrations. We recommend using Power Automate workflows to provide maximum security for your data. For more information, see Retirement of Office 365 connectors within Microsoft Teams.

Azure DevOps Services

> **Note**
>
> This feature is only available on Azure DevOps Services. Typically, new features are introduced in the cloud service first, and then made available on-premises in the next major version or update of Azure DevOps Server. For more information, see Azure DevOps Feature Timeline.

This article shows you how to use the Azure Pipelines app for Microsoft Teams to monitor pipeline events. You can set up and get notifications in your Teams channel for pipeline builds, releases, and approvals. Approvers can also approve releases from within the Teams channel.

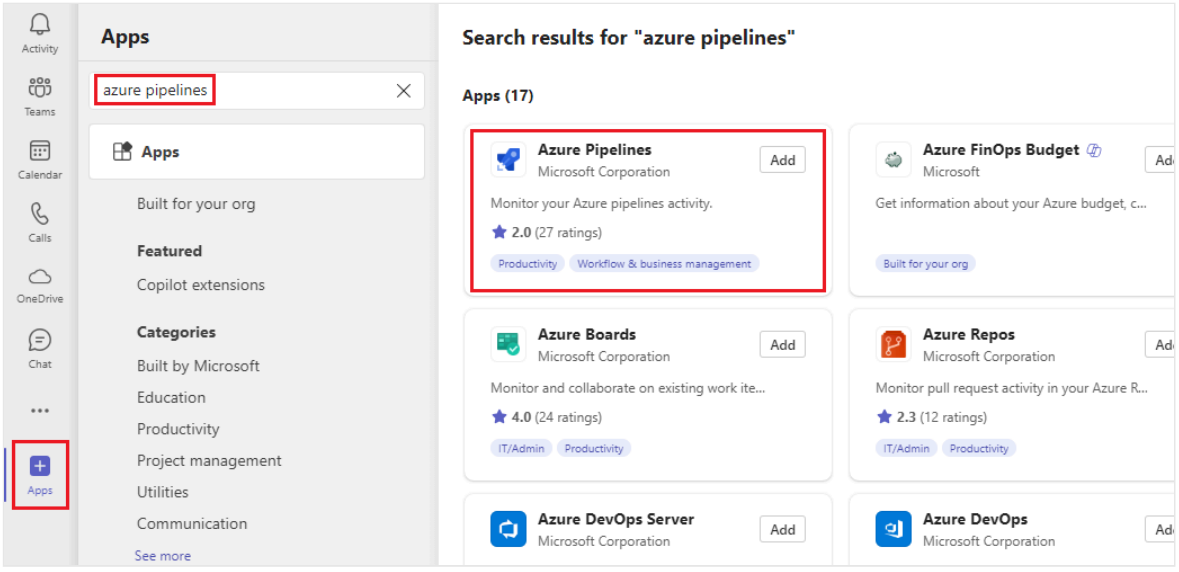
> **Note**
>
> Azure Pipelines notifications aren't supported inside Teams chat or direct messages.

Prerequisites

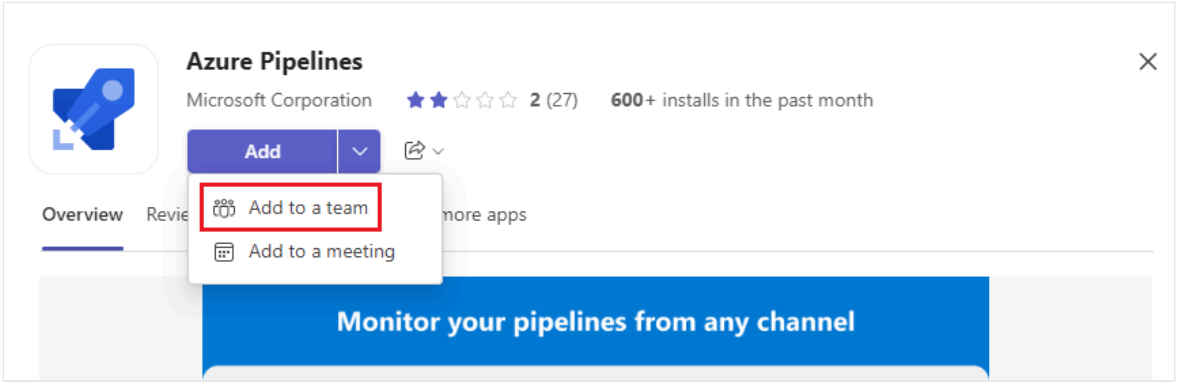
- Access to a Team in Microsoft Teams where you can add an app.
- Project Administrator or Build Administrator permissions in an Azure DevOps project. For more information, see Create a project and Pipeline security resources.
- **Third party application access via OAuth** enabled in Azure DevOps organizational settings.

Set up the Azure Pipelines app

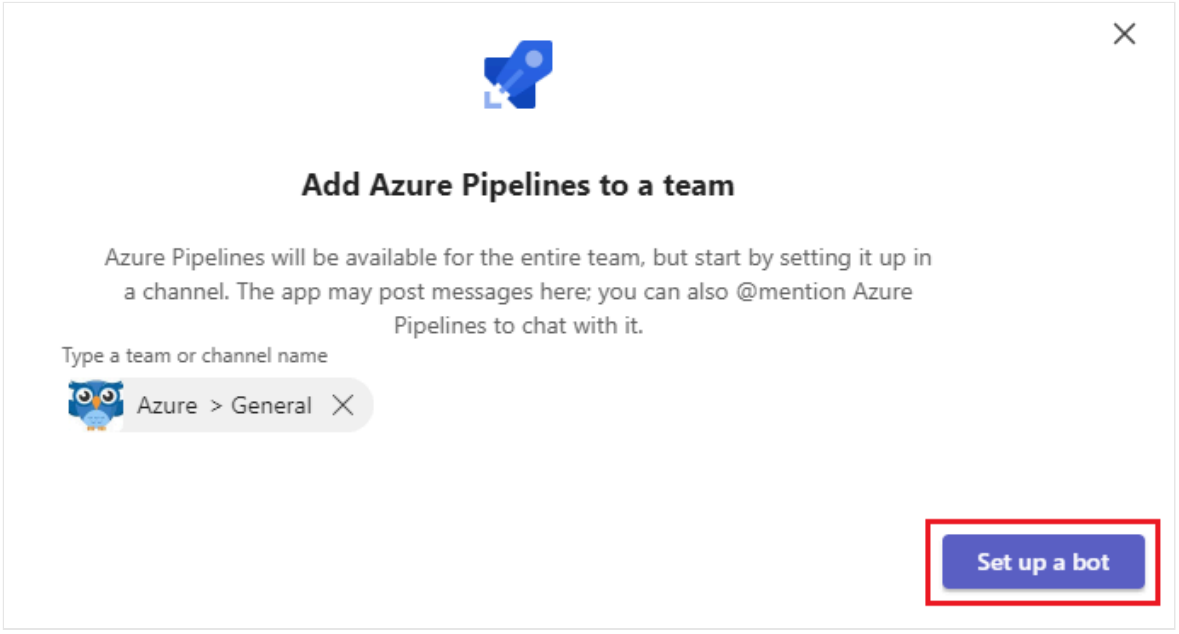
1. In Microsoft Teams, select **Apps**, search for *Azure Pipelines*, and then select **Azure Pipelines**.



2. Select the dropdown arrow next to **Add**, and select **Add to a team**.



3. Select or enter your team name, and then select **Set up a bot**.



Use Azure Pipelines app commands

The Azure Pipelines Teams app supports the following commands:

Expand table

Slash command	Functionality
@azure pipelines signin	Sign in to your Azure Pipelines account.
@azure pipelines signout	Sign out from your Azure Pipelines account.
@azure pipelines subscribe <pipeline url | project url>	Subscribe to a pipeline or all pipelines in a project to receive notifications.
@azure pipelines subscriptions	View, add, or remove subscriptions for this channel.
@azure pipelines unsubscribe all <project url>	Remove all pipelines belonging to a project and their associated subscriptions from a channel.
@azure pipelines help	Get help on the commands.
@azure pipelines feedback	Report a problem or suggest a feature.

Sign in to Azure Pipelines

1. In the Teams conversation pane, enter `@azurePipelines signin`.
2. Select **Sign in** and complete authentication to Azure Pipelines.

Subscribe to pipelines

Use the following commands to subscribe to and monitor all pipelines in a project or only specific pipelines.

- **All pipelines in a project:** The URL can be to your project or any page within your project, except to a pipeline. For example:

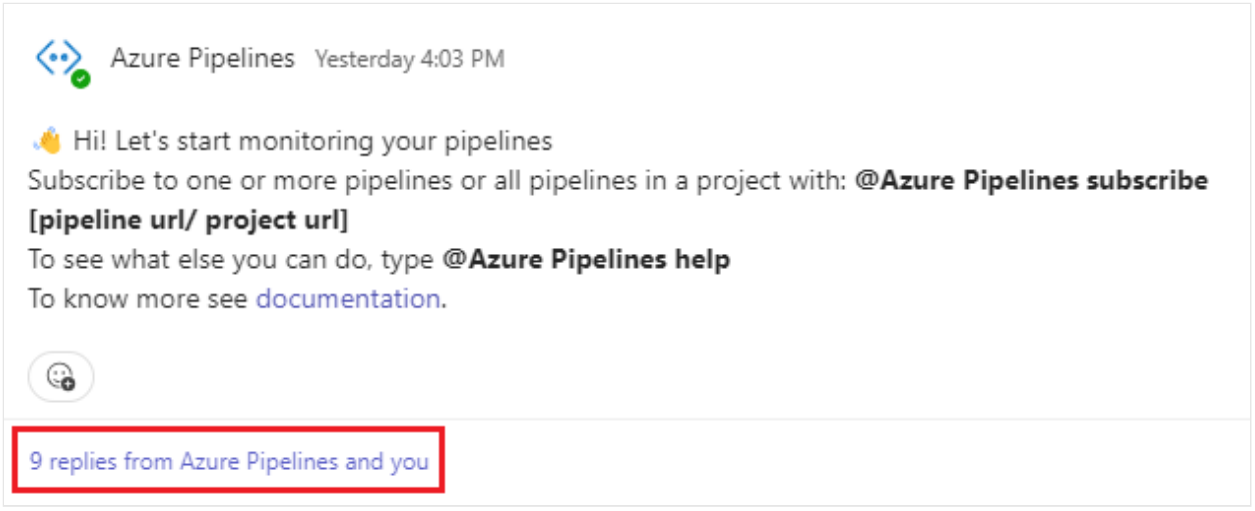
```
@azure pipelines subscribe https://dev.azure.com/myorg/myproject/
```

- **A specific pipeline:** The pipeline URL can be to any page within a pipeline that has a `definitionId` or `buildId/releaseId` in the URL. For example:

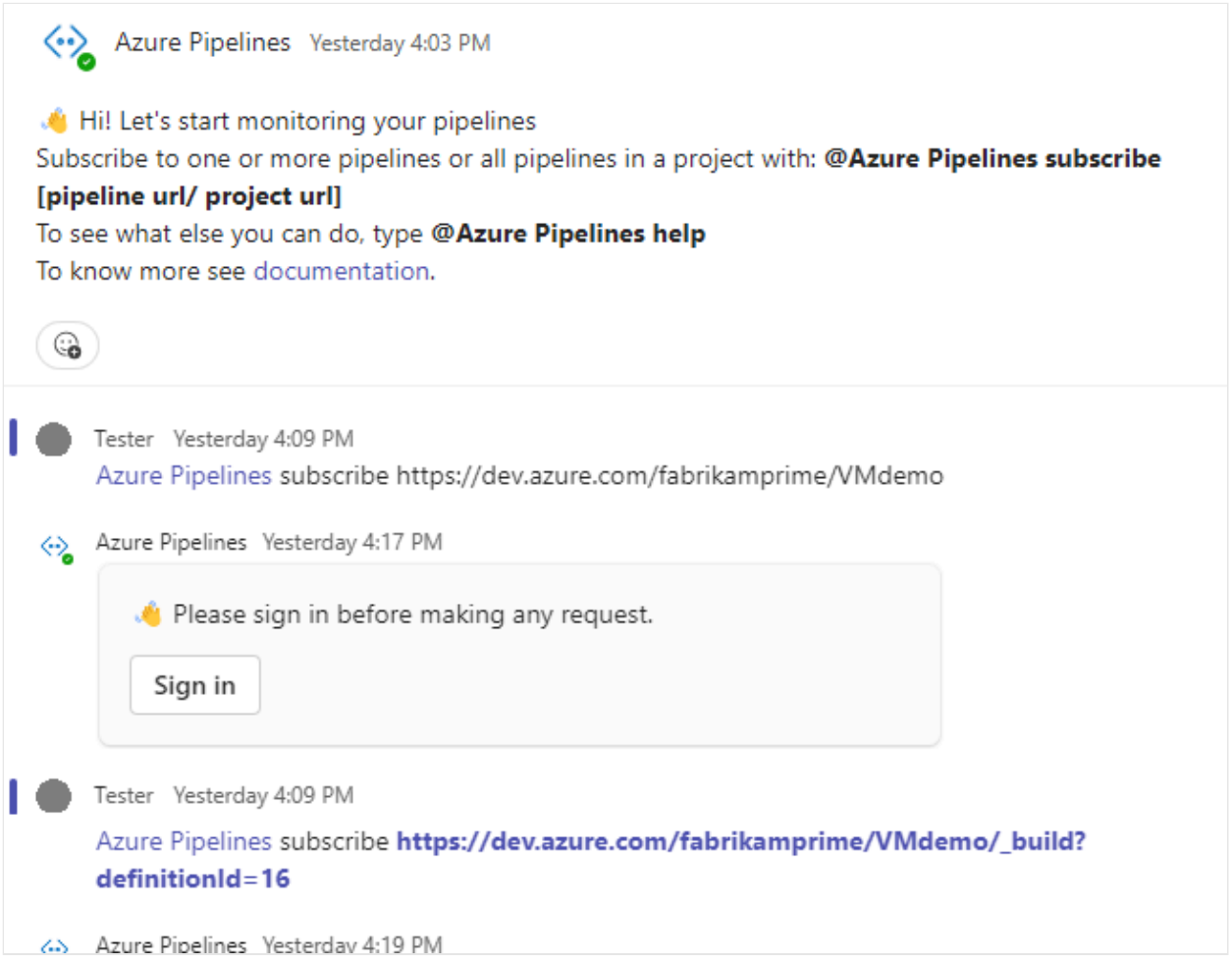
```
@azure pipelines subscribe
https://dev.azure.com/myorg/myproject/_build?definitionId=123
```

Expand linked notifications

All replies for a particular post are linked together.



To expand the thread, select the compacted thread link.



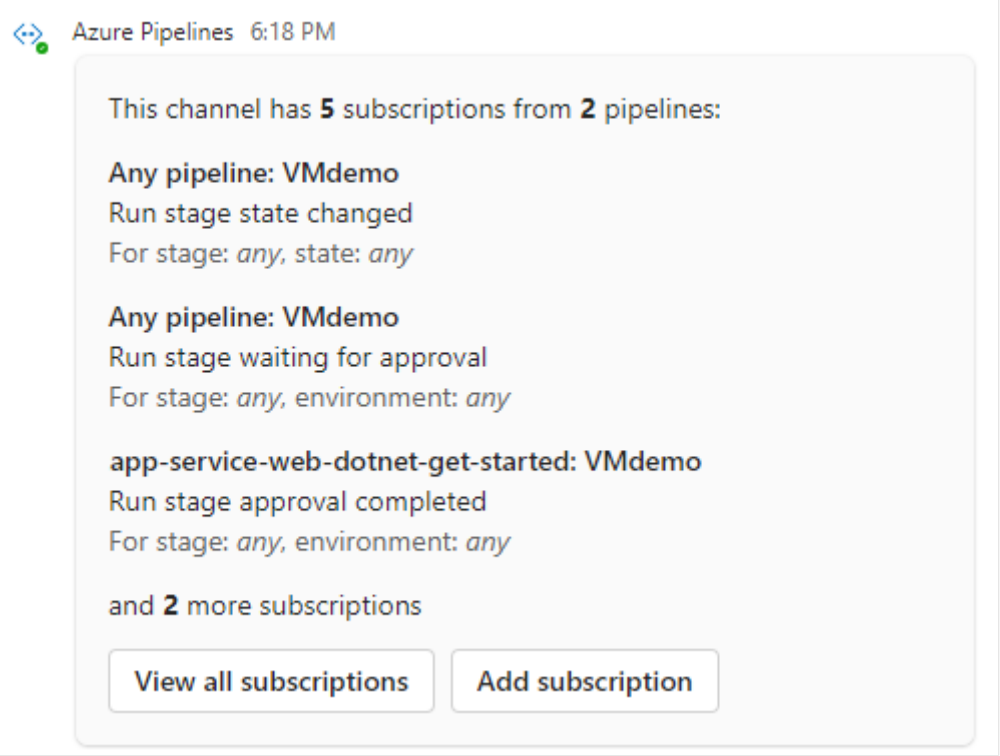
Manage subscriptions

When you subscribe to a pipeline, a few subscriptions get created by default without any filters applied. These subscriptions include **Run state changed** and **Run stage waiting for approval** for YAML pipelines, and **Release deployment approval pending** for Classic releases. You can remove these subscriptions or add more subscriptions.

The Azure Pipelines app also supports filters to customize what you see in your channel. For example, you might want to get notified only when builds fail or when deployments get pushed to a production environment.

To manage your subscriptions, complete the following steps.

1. To list all pipelines subscriptions, run the `@azure pipelines subscriptions` command.



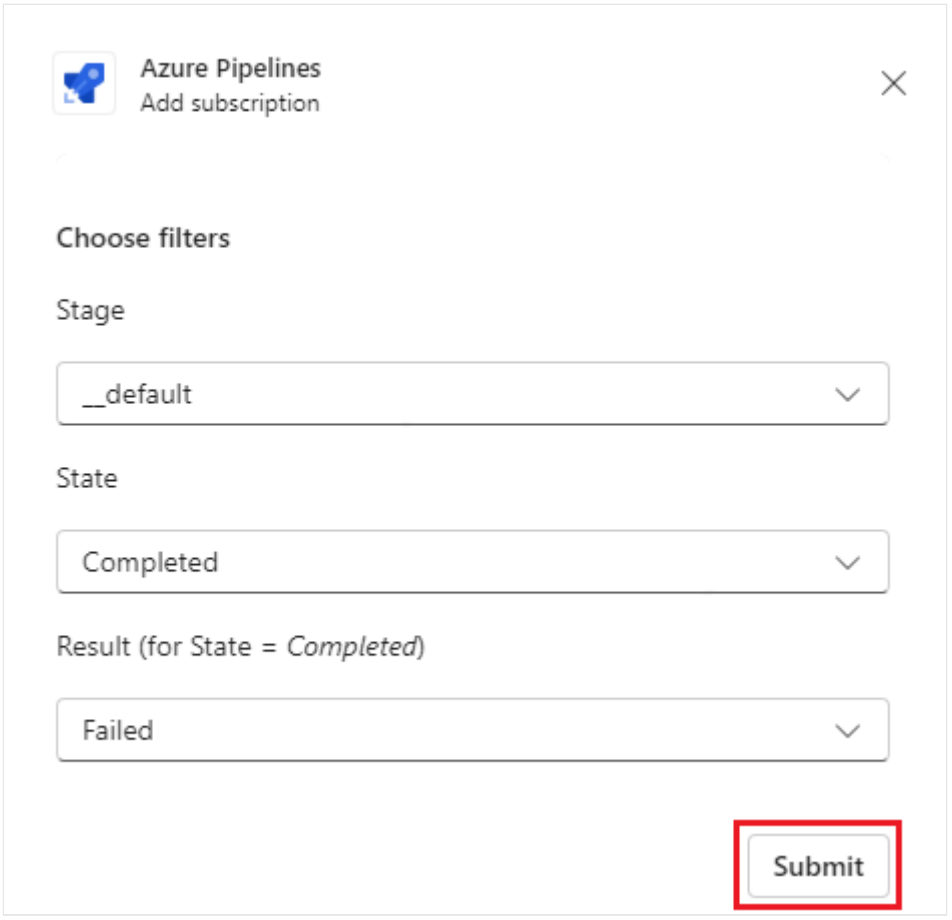
2. To remove a subscription, select **View all subscriptions**. Select **Remove** under any subscription you don't want, and then select **OK**.

3. To add a subscription, select **Add subscription**.

4. Select the event and the pipeline you want to subscribe to, and select **Next**.

5. Choose any **Stage** and **Environment** filters you want, select **Submit**, and then select **OK**.

   For example, the following subscription provides notifications for the **_default** stage only when the **Completed** state is **Failed**.

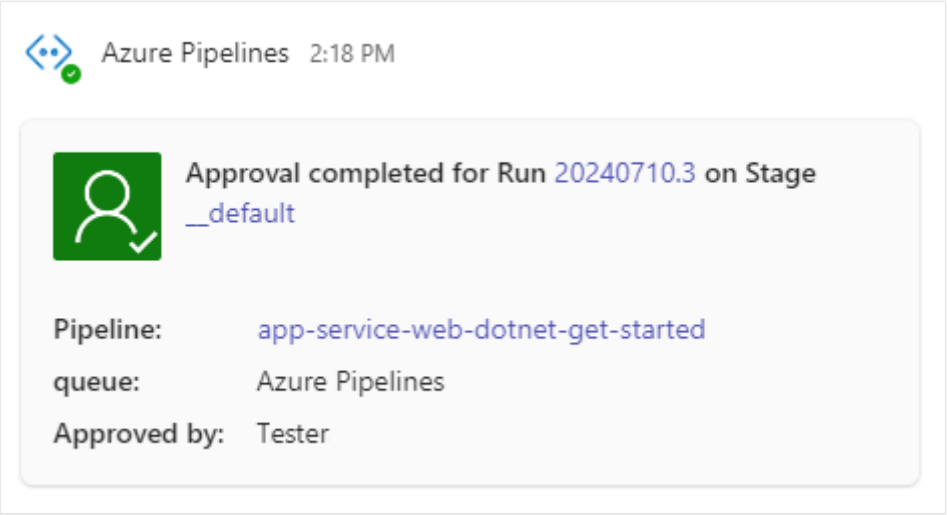


> **Note**
>
> Team Administrators can't remove or modify subscriptions that are created by Project Administrators.

See approval notifications

To see approval notifications, make sure you subscribe to the **Run stage waiting for approval** notification for YAML pipelines or the **Release deployment approval pending** notification for Classic releases. These subscriptions are created by default when you subscribe to the pipeline.

If you subscribe to the **Run stage approval completed** notification, you can also see when the stage is approved.

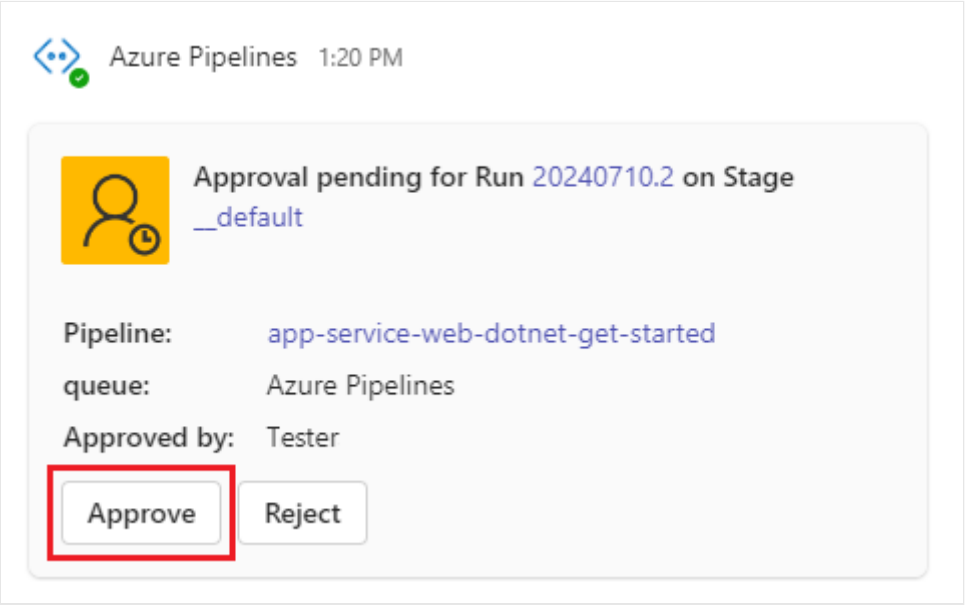


> **Note**
>
> You can't subscribe to deployment approvals that have the **Revalidate identity of approver before completing the approval** policy applied.

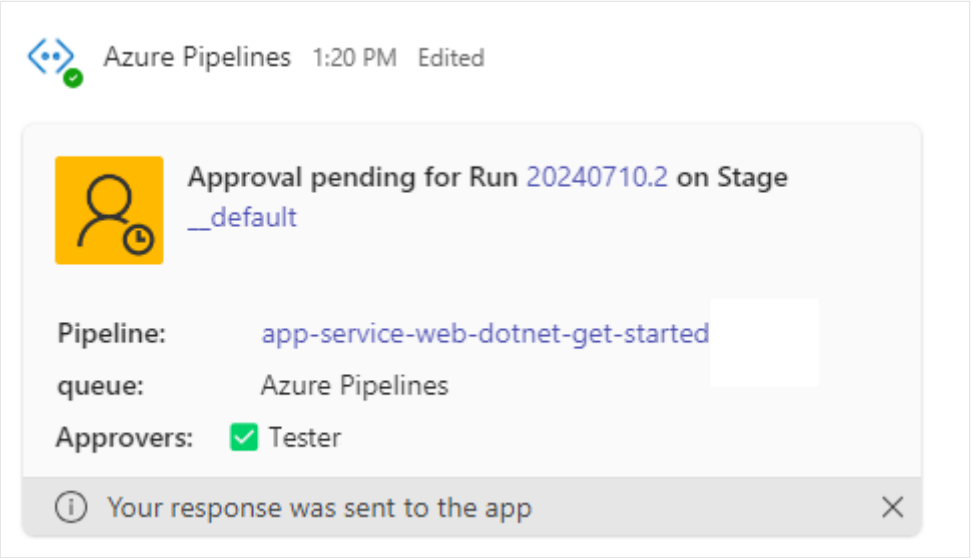
Approve from your channel

If you're an approver, you can approve deployments from within your Teams channel. The Azure Pipelines app supports all Azure Pipelines checks and approval scenarios. You can approve requests as an individual or for a team.

Whenever the running of a stage is pending your approval, the app posts a notification card with options to **Approve** or **Reject** the request in the channel. You can review the details of the request in the notification and take appropriate action.



The response is sent to the app.

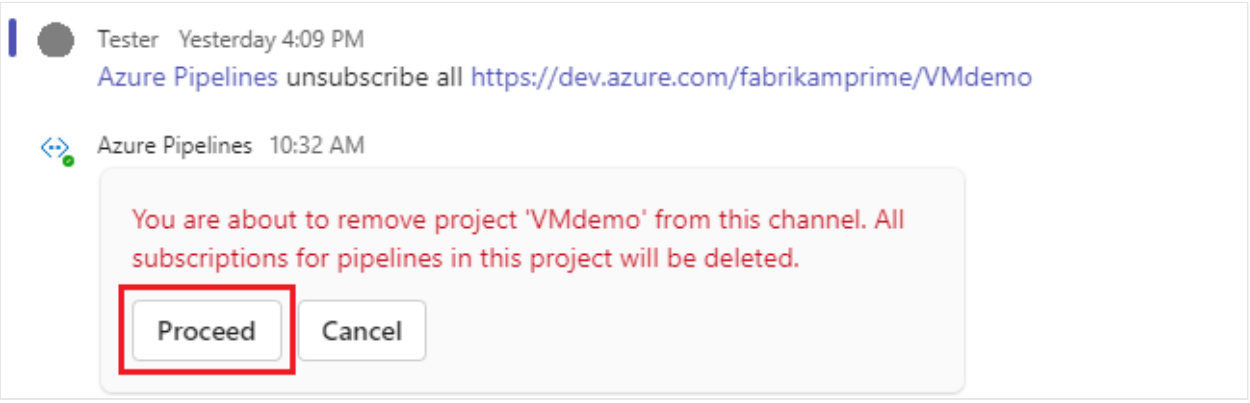


If you subscribed to **Run stage approval completed** notifications, you can also see when the stage is approved.

Unsubscribe from a channel

Run the `unsubscribe` command to delete all the subscriptions related to any pipeline in the project and remove the pipelines from the channel. For example:

```
@azure pipelines unsubscribe all https://dev.azure.com/myorg/myproject
```



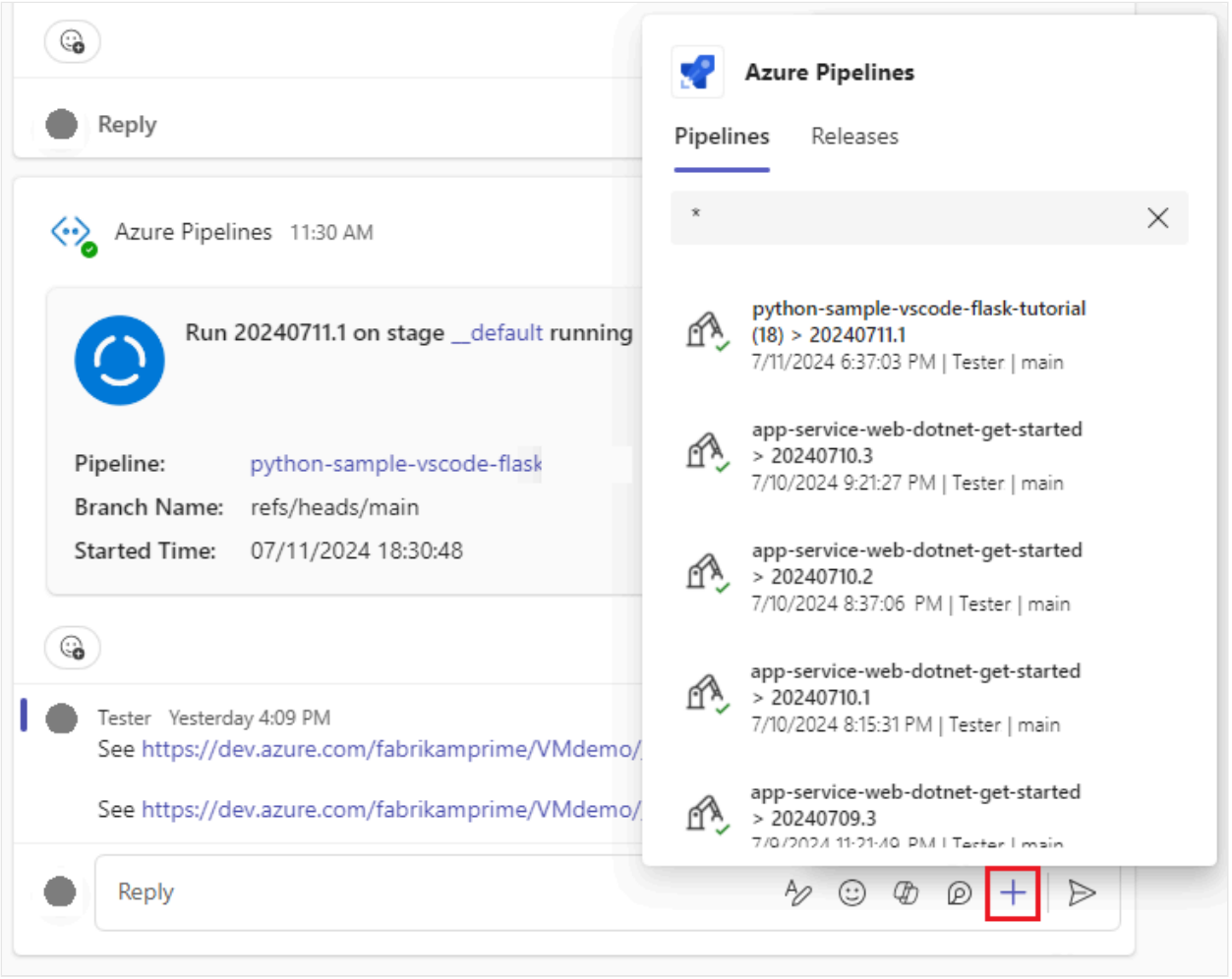
To delete the project and all subscriptions from the channel, select **Proceed**.

> **Important**
>
> Only Project Administrators can run this command.

Use the compose extension

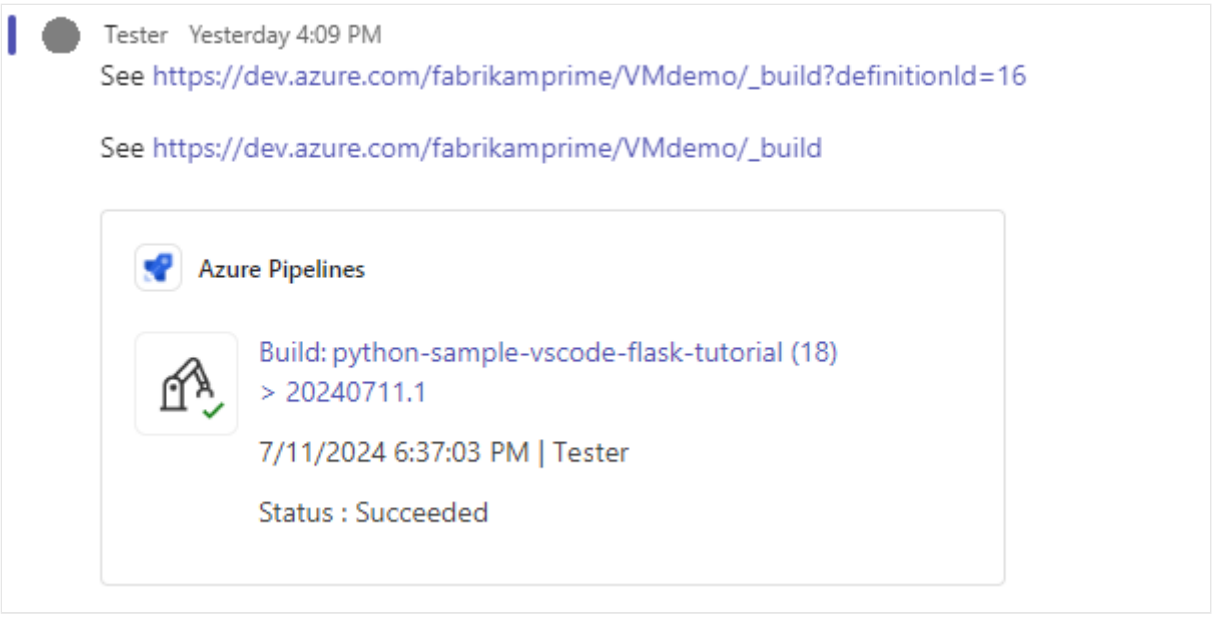
To help you search and share information about pipelines, the Azure Pipelines app for Teams supports a compose extension in messages. You can use the extension to search for pipelines in a project by pipeline ID or pipeline name.

To use the extension, you must be signed in to the Azure Pipelines project in the Teams channel. Select the + symbol in the message field, select **Azure Pipelines**, and then search for your pipeline or release.

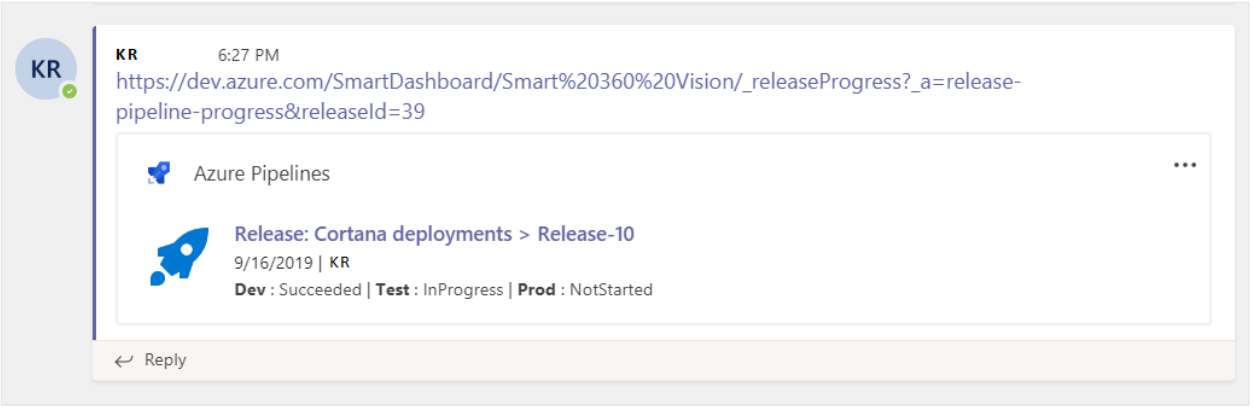


Preview of pipeline URLs

When you use the compose extension to add a pipeline URL to a Teams message, you see a preview similar to the following images. The preview helps to keep pipeline-related conversations relevant and up to date.



The following example shows a Release URL preview:



Connect multiple tenants

If you use different emails or tenants for Microsoft Teams and Azure DevOps, follow these steps to sign in and connect based on your settings.

Expand table

Microsoft Teams	Azure DevOps	Sign in action
email1@abc.com (tenant1)	email1@abc.com (tenant1)	Select **Sign in**
email1@abc.com (tenant1)	email2@pqr.com (tenant2)	1. Sign in to Azure DevOps. 2. In the same browser, start a new tab and go to https://teams.microsoft.com/. 3. Run the `sign in` command and select **Sign in**.

Microsoft Teams	Azure DevOps	Sign in action
email1@abc.com (tenant1)	email2@pqr.com (tenant2)	1. Select **Sign in with different email address**. 2. In the email ID picker, use the email2 to sign in.
email1@abc.com (tenant1)	email2@pqr.com (nondefault tenant3)	Not supported.

Troubleshoot authentication issues

In the **same browser**, start a new tab and sign in to `https://teams.microsoft.com/`. Run the `@Azure Pipelines signout` command and then run the `@Azure Pipelines signin` command in the channel where the Azure Pipelines app for Microsoft Teams is installed.

Select the **Sign in** button, and complete the sign-in process. Ensure that the directory shown is the same as what you chose in the previous step.

If these steps don't resolve your authentication issue, reach out to the Developer Community.

Related articles

- Use Azure Boards with Microsoft Teams
- Use Azure Repos with Microsoft Teams

Feedback

Was this page helpful? Yes No

Provide product feedback

Azure DevOps Services REST API Reference

Article • 03/31/2023

Welcome to the Azure DevOps Services/Azure DevOps Server REST API Reference.

Representational State Transfer (REST) APIs are service endpoints that support sets of HTTP operations (methods), which provide create, retrieve, update, or delete access to the service's resources. This article walks you through:

- The basic components of a REST API request/response pair.
- Overviews of creating and sending a REST request, and handling the response.

> Most REST APIs are accessible through our client libraries, which can be used to greatly simplify your client code.

Components of a REST API request/response pair

A REST API request/response pair can be separated into five components:

1. The **request URI**, in the following form: `VERB https://{instance}[/{team-project}]/_apis[/{area}]/{resource}?api-version={version}`

    - *instance*: The Azure DevOps Services organization or TFS server you're sending the request to. They are structured as follows:
        - Azure DevOps Services: `dev.azure.com/{organization}`
        - TFS: `{server:port}/tfs/{collection}` (the default port is 8080, and the value for collection should be `DefaultCollection` but can be any collection)
    - *resource path*: The resource path is as follows: `_apis/{area}/{resource}`. For example `_apis/wit/workitems`.
    - *api-version*: Every API request should include an api-version to avoid having your app or service break as APIs evolve. api-versions are in the following format: `{major}.{minor}[-{stage}[.{resource-version}]]`, for example:
        - `api-version=1.0`
        - `api-version=1.2-preview`
        - `api-version=2.0-preview.1`

> Note: *area* and *team-project* are optional, depending on the API request. Check out the TFS to REST API version mapping matrix below to find which REST API versions apply to your version of TFS.

2. HTTP **request message header** fields:

   - A required HTTP method (also known as an operation or verb), which tells the service what type of operation you are requesting. Azure REST APIs support GET, HEAD, PUT, POST, and PATCH methods.
   - Optional additional header fields, as required by the specified URI and HTTP method. For example, an Authorization header that provides a bearer token containing client authorization information for the request.

3. Optional HTTP **request message body** fields, to support the URI and HTTP operation. For example, POST operations contain MIME-encoded objects that are passed as complex parameters.

   - For POST or PUT operations, the MIME-encoding type for the body should be specified in the Content-type request header as well. Some services require you to use a specific MIME type, such as `application/json`.

4. HTTP **response message header** fields:

   - An HTTP status code, ranging from 2xx success codes to 4xx or 5xx error codes. Alternatively, a service-defined status code may be returned, as indicated in the API documentation.
   - Optional additional header fields, as required to support the request's response, such as a `Content-type` response header.

5. Optional HTTP **response message body** fields:

   - MIME-encoded response objects may be returned in the HTTP response body, such as a response from a GET method that is returning data. Typically, these objects are returned in a structured format such as JSON or XML, as indicated by the `Content-type` response header. For example, when you request an access token from Azure AD, it will be returned in the response body as the `access_token` element, one of several name/value paired objects in a data collection. In this example, a response header of `Content-Type: application/json` is also included.

Create the request

Authenticate

There are many ways to authenticate your application or service with Azure DevOps Services or TFS. The following table is an excellent way to decide which method is the best for you:

Expand table

Type of application	Description	example	Authentication mechanism	Code samples
Interactive client-side	Client application, that allows user interaction, calling Azure DevOps Services REST APIs	Console application enumerating projects in an organization	Microsoft Authentication Library (MSAL)	sample
Interactive JavaScript	GUI based JavaScript application	AngularJS single page app displaying project information for a user	MSAL	sample
Non-interactive client-side	Headless text only client side application	Console app displaying all bugs assigned to a user	Device Profile	sample
Interactive web	GUI based web application	Custom Web dashboard displaying build summaries	OAuth	sample
TFS application	TFS app using the Client OM library	TFS extension displaying team bug dashboards	Client Libraries	sample
Azure DevOps Services Extension	Azure DevOps Services extension	Azure DevOps extension samples	VSS Web Extension SDK	sample walkthrough

> **Note:** You can find more information on authentication on our authentication guidance page.

Assemble the request

Azure DevOps Services

For Azure DevOps Services, *instance* is `dev.azure.com/{organization}`, so the pattern looks like this:

```
VERB https://dev.azure.com/{organization}/_apis[/{area}]/{resource}?api-version={version}
```

For example, here's how to get a list of team projects in a Azure DevOps Services organization.

dos

```
curl -u {username}[:{personalaccesstoken}]
https://dev.azure.com/{organization}/_apis/projects?api-version=2.0
```

If you wish to provide the personal access token through an HTTP header, you must first convert it to a Base64 string (the following example shows how to convert to Base64 using C#). (Some tools apply a Base64 encoding by default. If you are trying the API via such tools, Base64 encoding of the PAT is not required) The resulting string can then be provided as an HTTP header in the format:

```
Authorization: Basic BASE64PATSTRING
```

Here it is in C# using the [HttpClient class](/previous-versions/visualstudio/hh193681(v=vs.118).

cs

```
public static async void GetProjects()
{
    try
    {
        var personalaccesstoken = "PAT_FROM_WEBSITE";

        using (HttpClient client = new HttpClient())
        {
            client.DefaultRequestHeaders.Accept.Add(
                new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"))
;

            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic",
                Convert.ToBase64String(
```

```
                    System.Text.ASCIIEncoding.ASCII.GetBytes(
                        string.Format("{0}:{1}", "",
personalaccesstoken))));

            using (HttpResponseMessage response = await client.GetAsync(

"https://dev.azure.com/{organization}/_apis/projects"))
            {
                response.EnsureSuccessStatusCode();
                string responseBody = await
response.Content.ReadAsStringAsync();
                Console.WriteLine(responseBody);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Most samples on this site use Personal Access Tokens as they're a compact example for authenticating with the service. However, there are a variety of authentication mechanisms available for Azure DevOps Services including MSAL, OAuth and Session Tokens. Refer to the Authentication section for guidance on which one is best suited for your scenario.

TFS

For TFS, `instance` is `{server:port}/tfs/{collection}` and by default the port is 8080. The default collection is `DefaultCollection`, but can be any collection.

Here's how to get a list of team projects from TFS using the default port and collection.

dos

```
curl -u {username}[:{personalaccesstoken}]
https://{server}:8080/tfs/DefaultCollection/_apis/projects?api-version=2.0
```

The examples above use personal access tokens, which requires that you create a personal access token.

Process the response

You should get a response like this.

JSON

```
{
    "value": [
        {
            "id": "eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "name": "Fabrikam-Fiber-TFVC",
            "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "description": "TeamFoundationVersionControlprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "collectionUrl": "https: //dev.azure.com/fabrikam-fiber-
inc/DefaultCollection"
            },
            "defaultTeam": {
                "id": "66df9be7-3586-467b-9c5f-425b29afedfd",
                "name": "Fabrikam-Fiber-TFVCTeam",
                "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1/teams/66df9be7-3586-
467b-9c5f-425b29afedfd"
            }
        },
        {
            "id": "6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "name": "Fabrikam-Fiber-Git",
            "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "description": "Gitprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "collectionUrl": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection"
            },
            "defaultTeam": {
                "id": "8bd35c5e-30bb-4834-a0c4-d576ce1b8df7",
                "name": "Fabrikam-Fiber-GitTeam",
                "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c/teams/8bd35c5e-30bb-
4834-a0c4-d576ce1b8df7"
            }
        }
    ],
    "count": 2
}
```


The response is JSON. That's generally what you'll get back from the REST APIs although there are a few exceptions, like Git blobs.

Now you should be able to look around the specific API areas like work item tracking or Git and get to the resources that you need. Keep reading to learn more about the general patterns that are used in these APIs.

API and TFS version mapping

Below you'll find a quick mapping of REST API versions and their corresponding TFS releases. All API versions will work on the server version mentioned as well as later versions.

Expand table

TFS Version	REST API Version	Build Version
Azure DevOps Server vNext	7.2	
Azure DevOps Server 2022.1	7.1	versions >= `19.225.34309.2`
Azure DevOps Server 2022	7.0	versions >= `19.205.33122.1`
Azure DevOps Server 2020	6.0	versions >= `18.170.30525.1`
Azure DevOps Server 2019	5.0	versions >= `17.143.28621.4`
TFS 2018 Update 3	4.1	versions >= `16.131.28106.2`
TFS 2018 Update 2	4.1	versions >= `16.131.27701.1`
TFS 2018 Update 1	4.0	versions >= `16.122.27409.2`
TFS 2018 RTW	4.0	versions >= `16.122.27102.1`
TFS 2017 Update 2	3.2	versions >= `15.117.26714.0`
TFS 2017 Update 1	3.1	versions >= `15.112.26301.0`
TFS 2017 RTW	3.0	versions >= `15.105.25910.0`
TFS 2015 Update 4	2.3	versions >= `14.114.26403.0`
TFS 2015 Update 3	2.3	versions >= `14.102.25423.0`
TFS 2015 Update 2	2.2	versions >= `14.95.25122.0`
TFS 2015 Update 1	2.1	versions >= `14.0.24712.0`
TFS 2015 RTW	2.0	versions >= `14.0.23128.0`

Related Content

Check out the Integrate documentation for REST API samples and use cases.

- Authentication guidance
- Samples

Client Libraries

Discover the client libraries for these REST APIs.

- .NET conceptual documentation and .NET reference documentation
- Go
- Node.js
- Python
- Swagger 2.0
- Web Extensions SDK

Where are the earlier versions of REST APIs? (Before 4.1)

We recently made a change to our engineering system and documentation generation process; we made this change to provide clearer, more in-depth, and more accurate documentation for everyone trying to use these REST APIs. Due to technical constraints, we are only able to document **API Version 4.1 and newer** using this method. We believe the documentation for API Version 4.1 and newer will be easier to use due to this change.

If you are working in TFS or are looking for the older versions of REST APIs, you can take a look at the REST API Overview for TFS 2015, 2017, and 2018.

Rate and usage limits

Article • 12/04/2023

Azure DevOps Services

Azure DevOps Services uses multi-tenancy to reduce costs and improve performance. This design leaves users vulnerable to performance issues and even outages when other users of their shared resources have spikes in their consumption. So, Azure DevOps limits the resources individuals can consume, and the amount of requests they can make to certain commands. When these limits are exceeded, future requests might be either delayed or blocked.

For more information, see Git limits and Best practices to avoid hitting rate limits.

Global consumption limit

Azure DevOps currently has a global consumption limit, which delays requests from individual users beyond a threshold when shared resources are in danger of being overwhelmed. This limit is focused exclusively on avoiding outages when shared resources are close to being overwhelmed. Individual users typically only get delayed requests when one of the following incidents occurs:

- One of their shared resources is at risk of being overwhelmed
- Their personal usage exceeds 200 times the consumption of a typical user within a (sliding) five-minute window

The amount of the delay depends on the user's sustained level of consumption. Delays range from a few milliseconds per request up to 30 seconds. Once consumption goes to zero or the resource is no longer overwhelmed, the delays stop within five minutes. If consumption remains high, delays might continue indefinitely to protect the resource.

When a user request gets delayed by a significant amount, that user receives an email and a warning banner in the web. For the build service account and others without an email address, members of the Project Collection Administrators group get the email. For more information, see Usage monitoring.

When an individual user's requests get blocked, responses with HTTP code 429 (too many requests) are received, with a message similar to the following message:

```
TF400733: The request has been canceled: Request was blocked due to exceeding usage of resource <resource name> in namespace <namespace ID>.
```

Azure DevOps throughput units (TSTUs)

Azure DevOps users consume many shared resources, and consumption depends on the following factors:

- Uploading a large number of files to version control creates a large amount of load on databases and storage accounts
- Complex work item tracking queries create database load based on the number of work items they search through
- Builds drive load by downloading files from version control, producing log output
- All operations consume CPU and memory on various parts of the service

To accommodate, Azure DevOps resource consumption is expressed in abstract units called Azure DevOps throughput units, or TSTUs. TSTUs eventually incorporate a blend of the following items:

- Azure SQL Database DTUs as a measure of database consumption
- Application tier and job agent CPU, memory, and I/O as a measure of compute consumption
- Azure Storage bandwidth as a measure of storage consumption

For now, TSTUs are primarily focused on Azure SQL Database DTUs, since Azure SQL Databases are the shared resources most commonly overwhelmed by excessive consumption. A single TSTU is the average load we expect a single normal user of Azure DevOps to generate per five minutes. Normal users also generate spikes in load. These spikes are typically 10 or fewer TSTUs per five minutes. Less frequently, spikes go as high as 100 TSTUs.

The global consumption limit is 200 TSTUs within a sliding five-minute window.

We recommend that you at least respond to the `Retry-After` header. If you detect a `Retry-After` header in any response, wait until some time passes before you send another request. Doing so helps your client application experience fewer enforced delays. Keep in mind that the response is 200, so you don't need to apply retry logic to the request.

If possible, we further recommend that you monitor `X-RateLimit-Remaining` and `X-RateLimit-Limit` headers. Doing so allows you to approximate how quickly you're approaching the delay threshold. Your client can intelligently react and spread out its requests over time.

> ⓘ **Note**

Identities that are used by tools and applications to integrate with Azure DevOps might need higher rate and usage limits beyond the allowed consumption limit from time to time. You can get additional rate and usage limits by assigning the **Basic + Test Plans** access level to the desired identities used by your application. Once the need for higher rate limits are fulfilled, you can go back to the access level that the identity used to have. You're charged for the cost of **Basic + Test Plans** access level only for the time it's assigned to the identity.

Identities that are already assigned a Visual Studio Enterprise subscription cannot be assigned **Basic + Test Plans** access level till they are removed.

Pipelines

Rate limiting is similar for Azure Pipelines. Each pipeline gets treated as an individual entity with its own resource consumption tracked. Even if build agents are self-hosted, they generate load in the form of cloning and sending logs.

We apply a 200 TSTU limit for an individual pipeline in a sliding 5-minute window. This limit is the same as the global consumption limit for users. If a pipeline gets delayed or blocked by rate limiting, a message appears in the attached logs.

API client experience

When requests get delayed or blocked, Azure DevOps returns response headers to help API clients react. While not fully standardized, these headers are broadly in line with other popular services.

The following table lists the headers available and what they mean. Except for `X-RateLimit-Delay`, all of these headers get sent before requests start getting delayed. This design gives clients the opportunity to proactively slow down their rate of requests.

Header name	Description
`Retry-After`	The RFC 6585-specified header sent to tell you how long to wait before you send your next request to fall under the detection threshold. Units: seconds.

`X-RateLimit-Resource`

A custom header indicating the service and type of threshold that was reached. Threshold types and service names might vary over time and without warning. We recommend displaying this string to a human, but not relying on it for computation.

`X-RateLimit-Delay`

How long the request was delayed. Units: seconds with up to three decimal places (milliseconds).

`X-RateLimit-Limit`

Total number of TSTUs allowed before delays are imposed.

`X-RateLimit-Remaining`

Number of TSTUs remaining before being delayed. If requests are already being delayed or blocked, it's 0.

`X-RateLimit-Reset`

Time at which, if all resource consumption stopped immediately, tracked usage would return to 0 TSTUs. Expressed in Unix epoch time.

Work tracking, process, & project limits

Azure DevOps imposes limits for the number of projects you can have in an organization and the number of teams you can have within each project. Also be aware of limits for work items, queries, backlogs, boards, dashboards, and more. For more information, see Work tracking, process, and project limits.

Wiki

In addition to the usual repository limits, wikis defined for a project are limited to 25 MB per single file.

Service connections

There are no per-project limits placed on creating service connections. However, there might be limits, which are imposed through Microsoft Entra ID. For additional information, review the following articles:

- Microsoft Entra service limits and restrictions
- Azure subscription and service limits, quotas, and constraints

Related articles

- Work tracking, process, and project limits
- Configure and customize Azure Boards
- Usage monitoring
- Git limits
- Best practices to avoid hitting rate limits

Deprecation of the Work Item Tracking (WIT) and Test Client OM

Article • 02/24/2023

Azure DevOps Services

WIT and Test Client OM are part of the broader TFS SDK. They are a set of SOAP-based APIs that can be used to create tools to run operations against the Azure DevOps. These SOAP-based APIs have been replaced by the set of modern REST-based endpoints. Therefore we are in the process of deprecating the SOAP-based APIs (Client OM) for WIT and Test.

> **Note**
>
> WIT and Test Client OM for Azure DevOps has been deprecated in latest version Visual Studio 2019 and the **Microsoft.TeamFoundationServer.ExtendedClient** package. This means, there's no new work item tracking or test functionality included into the Client OM.

Affected object models

Only the WIT and Test Client OM are being deprecated. Other areas like build and source control aren't on a deprecation schedule at this time.

Impact

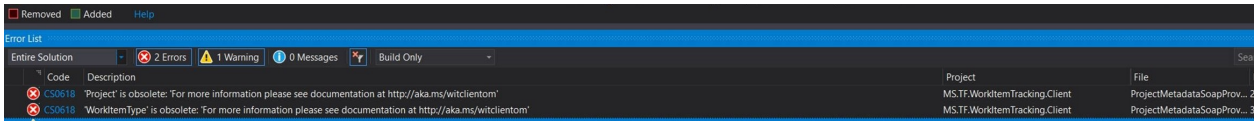
If your organization is using the WIT Client OM in tools that run against Azure DevOps, we recommend you start migrating those tools to stop using the WIT Client OM and start using the new REST based endpoints.

Updating your code to use REST is vital to ensure your tools work as we release new functionality into Azure Boards and Azure Test.

Visual Studio warnings

Any references to the latest versions of WIT Client OM result in an obsolete warning. This includes the dlls that come with Visual Studio 2019 and Azure DevOps 2019. Here is an example warning you may receive...

CS0618: 'WorkItemType' is obsolete: 'for more information, see documentation at https://aka.ms/witclientom '



Q&A

How do I fix my tools?

First thing you should investigate and determine if the tool is still needed. We have found that many organizations create a tool for a specific reason for a short period of time. Then for one reason or another, that tool continues to run when it's no longer needed. For this scenario, you can shut down the tool.

If the tool is still of use, you need to re-refactor that tool using the REST based .NET client libraries. Refactoring requires some reaction work on your part to remove references to the old WIT Client OM and replace them with the appropriate .NET Client Libraries.

We have also put together this handy migration guide with sample code to aid your transition from WIT Client OM to the REST-based .NET Client libraries.

Resources

- Migration Guide
- Azure DevOps Services REST API Reference
- .NET client libraries for Azure DevOps
- Understanding Rate limits
- Integration Best Practices
- Sample Code on GitHub

Migration guide

Article • 02/24/2023

Azure DevOps Services

This guide provides information to help migrate your .NET code from using WIT Client OM to our REST-based .NET Client Libraries. The following table lists common work tracking scenarios with links to their respected API documentation.

For sample code covered in these scenarios, see the GitHub Azure DevOps WIT Client OM Migration Guide.

Common scenarios

Scenario	WIT Client OM	REST based
Get list of work items	WorkItemStore.Query	Work Items - List
Get single work item	WorkItemStore.GetWorkItem	Work Items - Get Work Item
Create new work item	WorkItem	Work Items - Create
Update existing work item	WorkItem.Fields	Work Items - Update - Update a field
Validate a work item	WorkItem.IsValid(), WorkItem.Validate()	Work Items - Update - Validate only
Create a link to an existing work item	WorkItem.WorkItemLinks.Add	Work Item - Update - Add a link
Add a comment	WorkItem.History	Work Item - Update - Update a field
Create a hyperlink	WorkItem.Links.Add()	Work Item - Update - Add a hyperlink
Add an attachment	WorkItem.Attachments.Add()	Work Item - Update - Add an attachment
Query work items using WIQL	WorkItemStore.Query()	Wiql - Query by Wiql
Run an existing query to get a list of work items	WorkItemStore.Query()	Wiql - Query by ID

Scenario	WIT Client OM	REST based
Get list of work item types for your project	Category.WorkItemTypes	Work Item Types - List
Get work item type details	Category.WorkItemTypes	Work Item Types - Get
Get list of fields for a work item type	WorkItemType.FieldDefinitions	Work Item Types Field - List
Get field details	WorkItemType.FieldDefinitions	Work Item Types Field - Get

Resources

- Migrate data from Azure DevOps Server to Azure DevOps Services
- Migration guide with code samples
- Work Item Tracking REST API documentation

Support

Looking for a help on a scenario that we missed? If so, create a new issue on the migration guide project and we'll add the scenario to the repo.

Azure DevOps Services REST API Reference

Article • 03/31/2023

Welcome to the Azure DevOps Services/Azure DevOps Server REST API Reference.

Representational State Transfer (REST) APIs are service endpoints that support sets of HTTP operations (methods), which provide create, retrieve, update, or delete access to the service's resources. This article walks you through:

- The basic components of a REST API request/response pair.
- Overviews of creating and sending a REST request, and handling the response.

> Most REST APIs are accessible through our client libraries, which can be used to greatly simplify your client code.

Components of a REST API request/response pair

A REST API request/response pair can be separated into five components:

1. The **request URI**, in the following form: `VERB https://{instance}[/{team-project}]/_apis[/{area}]/{resource}?api-version={version}`

    - *instance*: The Azure DevOps Services organization or TFS server you're sending the request to. They are structured as follows:
        - Azure DevOps Services: `dev.azure.com/{organization}`
        - TFS: `{server:port}/tfs/{collection}` (the default port is 8080, and the value for collection should be `DefaultCollection` but can be any collection)
    - *resource path*: The resource path is as follows: `_apis/{area}/{resource}`. For example `_apis/wit/workitems`.
    - *api-version*: Every API request should include an api-version to avoid having your app or service break as APIs evolve. api-versions are in the following format: `{major}.{minor}[-{stage}[.{resource-version}]]`, for example:
        - `api-version=1.0`
        - `api-version=1.2-preview`
        - `api-version=2.0-preview.1`

> Note: *area* and *team-project* are optional, depending on the API request. Check out the TFS to REST API version mapping matrix below to find which REST API versions apply to your version of TFS.

2. HTTP **request message header** fields:

   - A required HTTP method (also known as an operation or verb), which tells the service what type of operation you are requesting. Azure REST APIs support GET, HEAD, PUT, POST, and PATCH methods.
   - Optional additional header fields, as required by the specified URI and HTTP method. For example, an Authorization header that provides a bearer token containing client authorization information for the request.

3. Optional HTTP **request message body** fields, to support the URI and HTTP operation. For example, POST operations contain MIME-encoded objects that are passed as complex parameters.

   - For POST or PUT operations, the MIME-encoding type for the body should be specified in the Content-type request header as well. Some services require you to use a specific MIME type, such as `application/json`.

4. HTTP **response message header** fields:

   - An HTTP status code, ranging from 2xx success codes to 4xx or 5xx error codes. Alternatively, a service-defined status code may be returned, as indicated in the API documentation.
   - Optional additional header fields, as required to support the request's response, such as a `Content-type` response header.

5. Optional HTTP **response message body** fields:

   - MIME-encoded response objects may be returned in the HTTP response body, such as a response from a GET method that is returning data. Typically, these objects are returned in a structured format such as JSON or XML, as indicated by the `Content-type` response header. For example, when you request an access token from Azure AD, it will be returned in the response body as the `access_token` element, one of several name/value paired objects in a data collection. In this example, a response header of `Content-Type: application/json` is also included.

Create the request

Authenticate

There are many ways to authenticate your application or service with Azure DevOps Services or TFS. The following table is an excellent way to decide which method is the best for you:

Expand table

Type of application	Description	example	Authentication mechanism	Code samples
Interactive client-side	Client application, that allows user interaction, calling Azure DevOps Services REST APIs	Console application enumerating projects in an organization	Microsoft Authentication Library (MSAL)	sample
Interactive JavaScript	GUI based JavaScript application	AngularJS single page app displaying project information for a user	MSAL	sample
Non-interactive client-side	Headless text only client side application	Console app displaying all bugs assigned to a user	Device Profile	sample
Interactive web	GUI based web application	Custom Web dashboard displaying build summaries	OAuth	sample
TFS application	TFS app using the Client OM library	TFS extension displaying team bug dashboards	Client Libraries	sample
Azure DevOps Services Extension	Azure DevOps Services extension	Azure DevOps extension samples	VSS Web Extension SDK	sample walkthrough

> **Note:** You can find more information on authentication on our authentication guidance page.

Assemble the request

Azure DevOps Services

For Azure DevOps Services, *instance* is `dev.azure.com/{organization}`, so the pattern looks like this:

```
VERB https://dev.azure.com/{organization}/_apis[/{area}]/{resource}?api-version={version}
```

For example, here's how to get a list of team projects in a Azure DevOps Services organization.

dos

```
curl -u {username}[:{personalaccesstoken}]
https://dev.azure.com/{organization}/_apis/projects?api-version=2.0
```

If you wish to provide the personal access token through an HTTP header, you must first convert it to a Base64 string (the following example shows how to convert to Base64 using C#). (Some tools apply a Base64 encoding by default. If you are trying the API via such tools, Base64 encoding of the PAT is not required) The resulting string can then be provided as an HTTP header in the format:

```
Authorization: Basic BASE64PATSTRING
```

Here it is in C# using the [HttpClient class](/previous-versions/visualstudio/hh193681(v=vs.118).

cs

```
public static async void GetProjects()
{
    try
    {
        var personalaccesstoken = "PAT_FROM_WEBSITE";

        using (HttpClient client = new HttpClient())
        {
            client.DefaultRequestHeaders.Accept.Add(
                new
System.Net.Http.Headers.MediaTypeWithQualityHeaderValue("application/json"))
;

            client.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Basic",
                Convert.ToBase64String(
```

```
                    System.Text.ASCIIEncoding.ASCII.GetBytes(
                        string.Format("{0}:{1}", "",
personalaccesstoken))));

            using (HttpResponseMessage response = await client.GetAsync(

"https://dev.azure.com/{organization}/_apis/projects"))
            {
                response.EnsureSuccessStatusCode();
                string responseBody = await
response.Content.ReadAsStringAsync();
                Console.WriteLine(responseBody);
            }
        }
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.ToString());
    }
}
```

Most samples on this site use Personal Access Tokens as they're a compact example for authenticating with the service. However, there are a variety of authentication mechanisms available for Azure DevOps Services including MSAL, OAuth and Session Tokens. Refer to the Authentication section for guidance on which one is best suited for your scenario.

TFS

For TFS, `instance` is `{server:port}/tfs/{collection}` and by default the port is 8080. The default collection is `DefaultCollection`, but can be any collection.

Here's how to get a list of team projects from TFS using the default port and collection.

dos

```
curl -u {username}[:{personalaccesstoken}]
https://{server}:8080/tfs/DefaultCollection/_apis/projects?api-version=2.0
```

The examples above use personal access tokens, which requires that you create a personal access token.

Process the response

You should get a response like this.

JSON

```
{
    "value": [
        {
            "id": "eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "name": "Fabrikam-Fiber-TFVC",
            "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1",
            "description": "TeamFoundationVersionControlprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https: //dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "collectionUrl": "https: //dev.azure.com/fabrikam-fiber-
inc/DefaultCollection"
            },
            "defaultTeam": {
                "id": "66df9be7-3586-467b-9c5f-425b29afedfd",
                "name": "Fabrikam-Fiber-TFVCTeam",
                "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/eb6e4656-77fc-42a1-9181-4c6d8e9da5d1/teams/66df9be7-3586-
467b-9c5f-425b29afedfd"
            }
        },
        {
            "id": "6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "name": "Fabrikam-Fiber-Git",
            "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c",
            "description": "Gitprojects",
            "collection": {
                "id": "d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "name": "DefaultCollection",
                "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projectCollections/d81542e4-cdfa-4333-b082-1ae2d6c3ad16",
                "collectionUrl": "https://dev.azure.com/fabrikam-fiber-
inc/DefaultCollection"
            },
            "defaultTeam": {
                "id": "8bd35c5e-30bb-4834-a0c4-d576ce1b8df7",
                "name": "Fabrikam-Fiber-GitTeam",
                "url": "https://dev.azure.com/fabrikam-fiber-
inc/_apis/projects/6ce954b1-ce1f-45d1-b94d-e6bf2464ba2c/teams/8bd35c5e-30bb-
4834-a0c4-d576ce1b8df7"
            }
        }
    ],
    "count": 2
}
```


The response is JSON. That's generally what you'll get back from the REST APIs although there are a few exceptions, like Git blobs.

Now you should be able to look around the specific API areas like work item tracking or Git and get to the resources that you need. Keep reading to learn more about the general patterns that are used in these APIs.

API and TFS version mapping

Below you'll find a quick mapping of REST API versions and their corresponding TFS releases. All API versions will work on the server version mentioned as well as later versions.

Expand table

TFS Version	REST API Version	Build Version
Azure DevOps Server vNext	7.2	
Azure DevOps Server 2022.1	7.1	versions >= `19.225.34309.2`
Azure DevOps Server 2022	7.0	versions >= `19.205.33122.1`
Azure DevOps Server 2020	6.0	versions >= `18.170.30525.1`
Azure DevOps Server 2019	5.0	versions >= `17.143.28621.4`
TFS 2018 Update 3	4.1	versions >= `16.131.28106.2`
TFS 2018 Update 2	4.1	versions >= `16.131.27701.1`
TFS 2018 Update 1	4.0	versions >= `16.122.27409.2`
TFS 2018 RTW	4.0	versions >= `16.122.27102.1`
TFS 2017 Update 2	3.2	versions >= `15.117.26714.0`
TFS 2017 Update 1	3.1	versions >= `15.112.26301.0`
TFS 2017 RTW	3.0	versions >= `15.105.25910.0`
TFS 2015 Update 4	2.3	versions >= `14.114.26403.0`
TFS 2015 Update 3	2.3	versions >= `14.102.25423.0`
TFS 2015 Update 2	2.2	versions >= `14.95.25122.0`
TFS 2015 Update 1	2.1	versions >= `14.0.24712.0`
TFS 2015 RTW	2.0	versions >= `14.0.23128.0`

Related Content

Check out the Integrate documentation for REST API samples and use cases.

- Authentication guidance
- Samples

Client Libraries

Discover the client libraries for these REST APIs.

- .NET conceptual documentation and .NET reference documentation
- Go
- Node.js
- Python
- Swagger 2.0
- Web Extensions SDK

Where are the earlier versions of REST APIs? (Before 4.1)

We recently made a change to our engineering system and documentation generation process; we made this change to provide clearer, more in-depth, and more accurate documentation for everyone trying to use these REST APIs. Due to technical constraints, we are only able to document **API Version 4.1 and newer** using this method. We believe the documentation for API Version 4.1 and newer will be easier to use due to this change.

If you are working in TFS or are looking for the older versions of REST APIs, you can take a look at the REST API Overview for TFS 2015, 2017, and 2018.

Settings & Usage documentation

Configure resources and manage settings for an organization, project, team, or user.

Get started

OVERVIEW

About settings

Get started as an administrator

QUICKSTART

Set user preferences

HOW-TO GUIDE

Enable preview features

Manage your organization (cloud)

OVERVIEW

About managing your organization

Plan your organizational structure

CONCEPT

Manage access with Microsoft Entra ID

QUICKSTART

Create an organization

HOW-TO GUIDE

Add users to your organization

Connect your organization to Microsoft Entra ID

Add & manage projects

CONCEPT

About projects & scaling up

Customize your project

OVERVIEW

Manage your project

HOW-TO GUIDE

Create a project

Connect to GitHub

Add & manage teams

CONCEPT

About teams & Agile tools

HOW-TO GUIDE

Add a team

Configure team tools

Define team area paths

Configure team iterations

Add a team administrator

Set alerts or notifications

CONCEPT

About notifications

HOW-TO GUIDE

Set personal notifications

Set team notifications

Audit & usage (cloud)

CONCEPT

Rate limits & usage

QUICKSTART

Access, export, and filter audit logs

Configure Pipelines resources

HOW-TO GUIDE

Create & manage agent pools

Create & manage deployment groups

Set build & release retention policies

Create & manage service connections

Configure Repos resources

OVERVIEW

About branches and branch policies

HOW-TO GUIDE

Create & manage Git repositories

Set Git repository settings and policies

Manage Git branch policies

Add TFVC check-in policies

Configure Test resources

HOW-TO GUIDE

Set test retention policies

Shipping Visual Studio Extensions

Article • 03/10/2023

After you've finished developing your extension, you can install it on other machines, share it with your friends and coworkers, or publish it on the Visual Studio Marketplace. In this section we explain all the things you need to do in order to publish and maintain your extension: working with .vsix files, publishing, localizing, and updating.

Working with VSIX Extensions

You can create a VSIX extensions by creating a blank VSIX Project, and then adding different item templates to it. For more information, see VSIX Project Template.

You can use the VSIX format to package project templates, item templates, VSPackages, Managed Extensibility Framework (MEF) components, **Toolbox** controls, assemblies, and custom types (this includes custom Start Pages for Visual Studio 2017). The VSIX format uses file-based deployment. For more information about VSIX packages, see Anatomy of a VSIX Package.

The VSIX format does not support the installation of code snippets. It also does not support certain other scenarios such as writing to the Global Assembly Cache (GAC) or to the system registry. If you need to write to the GAC or the registry in the installation, you must use the Windows Installer. For more information, see Preparing Extensions for Windows Installer Deployment.

Publishing your Extension to the Visual Studio Marketplace

You can distribute your extension to other people simply by mailing them the .vsix file or putting in on a server. But the best way to get your code in the hands of a lot of people is to put it on the Visual Studio Marketplace. Visual Studio Marketplace extensions are available to Visual Studio users through **Extensions and Updates**. For more information, see Finding and Using Visual Studio Extensions.

For a full example that shows how to upload an extension to the Visual Studio Marketplace, see Walkthrough: Publishing a Visual Studio Extension.

Private Galleries

As you develop controls, templates, and tools, you can share them with your organization by posting them to a private gallery on your intranet. For more information, see Private Galleries.

Localizing your Extension

If you're planning to release your extension in different locales, you should consider localizing it. For an explanation of what's involved, see Localizing VSIX Packages.

Updating and Versioning your Extension

After you've published your extension, there will come a time when you need to update it. To find out how to update an extension that has been published on the Visual Studio Marketplace, see How to: Update an Extension.

You can set your extension to support multiple versions of Visual Studio. For more information, see Supporting Multiple Versions of Visual Studio.

Related Topics

Expand table

Title	Description
Getting Started with the VSIX Project Template	Explains how to use the VSIX project template to install a custom project template.
Anatomy of a VSIX Package	Describes the components of a VSIX package.
VSIX Project Template	Provides step-by-step instructions about how to package and publish an extension.
Localizing VSIX Packages	Explains how to provide localized text for the installation process by using extension.vsixlangpack files.
How to: Update an Extension	Describes how to update an extension on your system and how to deploy an update to an existing Visual Studio extension.
How to: Add a Dependency to a VSIX Package	Describes how to add references to VSIX deployment packages.
Preparing Extensions for Windows Installer Deployment	Explains how to deploy your extension with Windows Installer.
Signing VSIX Packages	Explains how to sign VSIX packages.

Title	Description
Private Galleries	Explains how to create private galleries for extensions.
Supporting Multiple Versions of Visual Studio	Shows how to have your extension support multiple versions of Visual Studio.
Locating Visual Studio	Describes how to locate Visual Studio instances for custom extension deployment.

Windows app development documentation

Learn how to design, develop, and deploy apps and solutions for Windows PCs and other devices.

QUICKSTART
Start developing Windows apps

OVERVIEW
App framework options

OVERVIEW
Enhance your Windows apps with AI

OVERVIEW
Set up your environment with Dev Home, WinGet, Terminal, WSL, and more

REFERENCE
Windows APIs

DOWNLOAD
Download Visual Studio

DOWNLOAD
Windows Code Samples

TRAINING
Online training for Windows app developers

Dive into developing apps for Windows

Get started

Overview of framework options

Get started with WinUI

Windows App SDK and supported Windows releases

Design

Design and code your app UI

Design basics for Windows apps

Windows 11 design principles

Install tools for the Windows App SDK

Sample apps for Windows

Design toolkits

Develop

Windows development features

Windows App SDK features

WinUI

Windows Copilot Runtime

Deploy

Deployment architecture

Windows App SDK deployment guide

Package apps using MSIX

Manage your MSIX deployment

More development options

Windows Application SDK

.NET

UWP

Games

Contribute to Windows Open Source projects

Windows App SDK

A set of developer tools and APIs that represent the next evolution in the Windows app platform.

WinUI

The native user interface layer for building Windows apps including modern controls and styles.

MSIX SDK

.NET

Enables developers to pack and unpack app packages on a variety of platforms.

Engage with .NET open source projects including .NET, WPF, Windows Forms, and much more.

@WindowsDocs - OneDevMinute on YouTube - Windows developer support - Stack Overflow

About migrating and integrating work tracking data

Article • 02/10/2025

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

There are various tools available to help you migrate your work tracking data to the Azure DevOps platform. This article provides an overview of these tools and links to resources that support the migration of work tracking data and processes. You can also integrate Azure Boards with numerous non-Microsoft tools and extensions.

> **Note**
>
> With Azure DevOps Marketplace extensions for Azure Boards, you can customize and enhance the default experience. For more information, see the list of recommended extensions for Azure Boards, further in this article.

Migrate from Azure DevOps Server

The Data Migration Tool for Azure DevOps provides a high fidelity way to migrate collection databases from Azure DevOps Server to Azure DevOps Services. For more information, see Migrate data from Azure DevOps Server to Azure DevOps Services.

Migrate data between projects

You can search for extensions in the Visual Studio Marketplace that allow you to bulk edit and migrate data between project on both on-premises and the cloud. Such tools generally support the following tasks:

- Migrate work items from one project to another project and synchronize changes after a migration
- Merge many projects into a single project
- Split one project into many projects
- Assist changing process templates
- Edit work items in bulk
- Migrate test suites and test plans

Migrate process models between Azure DevOps organizations

For constraints on process template definitions that you can import, see Resolve validation errors for process import.

Export and import work tracking data

The main tool you can use to import work tracking data you exported from elsewhere is Microsoft Excel. Excel supports publishing a flat list of work items or a hierarchical tree of parent-child linked work items. For more information, see Bulk add or modify work items with Excel.

Integrate with GitHub

Azure Boards (cloud) and Azure DevOps Server integrate with GitHub. Connect Azure Boards with GitHub repositories to link between GitHub commits, pull requests, and issues to work items. You can use GitHub for software development while using Azure Boards to plan and track your work.

For more information, see Connect Azure Boards to GitHub.

Integrate with non-Microsoft tools using service hooks

You can integrate Azure Boards with other non-Microsoft tools such as Jenkins and Trello. Using service hooks you can generate actions based on Azure DevOps events. For more information, see the following articles:

- Create a service hook for Azure DevOps with Jenkins
- Create a service hook for Azure DevOps with Trello
- Integrate with service hooks

Extensions for Azure Boards

The following table contains some Microsoft extensions. You can search for other non-Microsoft extensions in the Visual Studio Marketplace.

Expand table

Category	Extensions
Automation	Power Automate, Azure DevOps
Command-line interface	Azure DevOps CLI
Customizing work item types	- Cascading Lists - Color picklist control - Multi-value control - Work Item Visualization - WSJF (Weighted Shortest Job First)
Dashboard widgets	- Azure Application Insights widget - Work Item Details widget - Roll-up Board widget
Product planning	- Azure DevOps Open in Excel - Epic Roadmap extension, Feature Timeline (Deprecated) - Feature Timeline extension, Feature Timeline (Deprecated) - Retrospectives - Split! - Team Calendar - Reactivations Report
Querying and reporting	WIQL to OData

Related articles

- Bulk add or modify work items with Microsoft Excel
- Use the Azure DevOps REST API

Feedback

Was this page helpful? Yes No

Provide product feedback

Find, install, and manage extensions for Visual Studio

Article • 09/05/2024

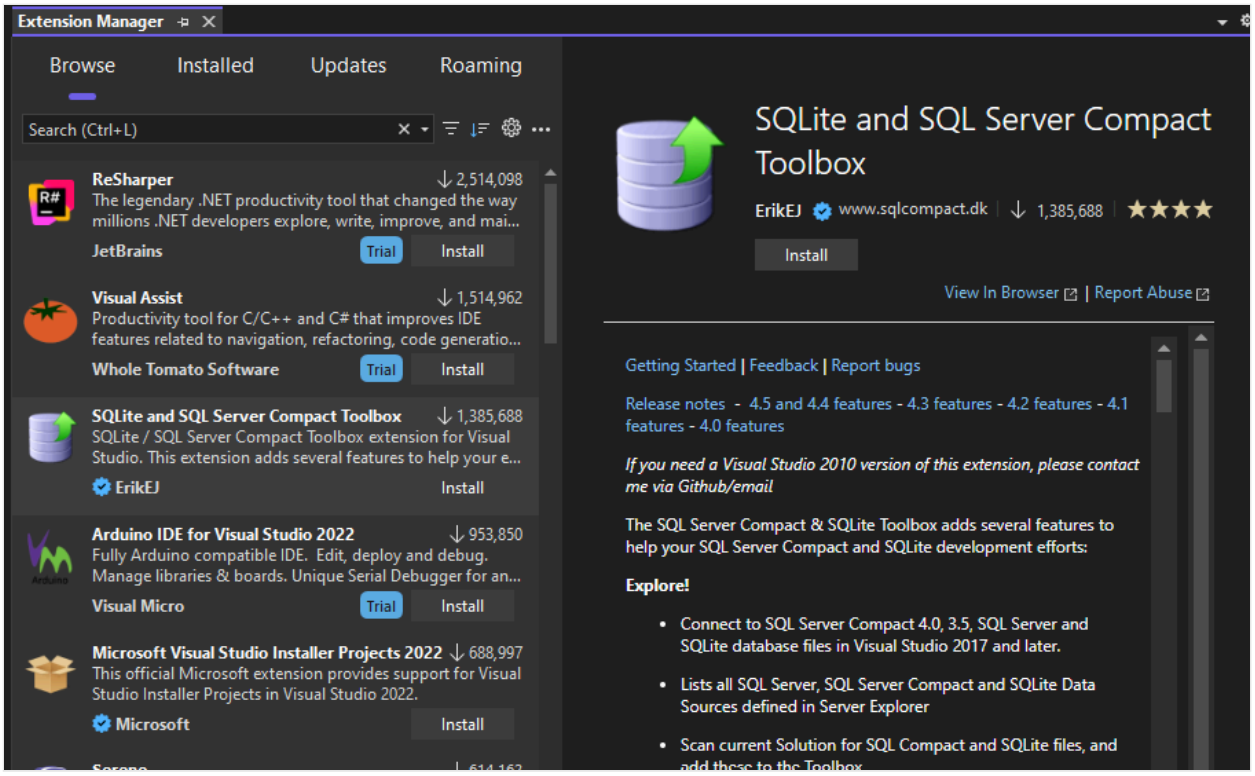
This article describes how to find, install, and manage extension packages in Visual Studio. Extensions are code packages that run inside Visual Studio and provide new or improved features. Extensions can be controls, samples, templates, tools, or other components that add functionality to Visual Studio, for example, Live Share or GitHub Copilot.

For information about creating Visual Studio extensions, see Visual Studio SDK. For information about using extensions, see the individual extension page on Visual Studio Marketplace. For information about finding extensions, see the Where Are My Favorite Extensions in Visual Studio 2022? blog post.

> **Note**
>
> To learn more about a new extensibility model that's currently in preview, see the Build Better Extensions with VisualStudio.Extensibility (Preview 3) blog post.

Use Extension Manager

In the Visual Studio IDE, Extension Manager is the tool you use to find, install, and manage Visual Studio extensions. To open Extension Manager, select **Extensions** > **Manage Extensions**. Or type **extensions** in the search box and select **Manage Extensions**.



The left pane categorizes extensions by those that are available on Visual Studio Marketplace (**Browse**), those that are installed, and those that have updates available. The **Roaming** tab lists all the Visual Studio extensions that you have installed on any machine or instance of Visual Studio. It's designed to let you find your favorite extensions more easily.

Find and install extensions

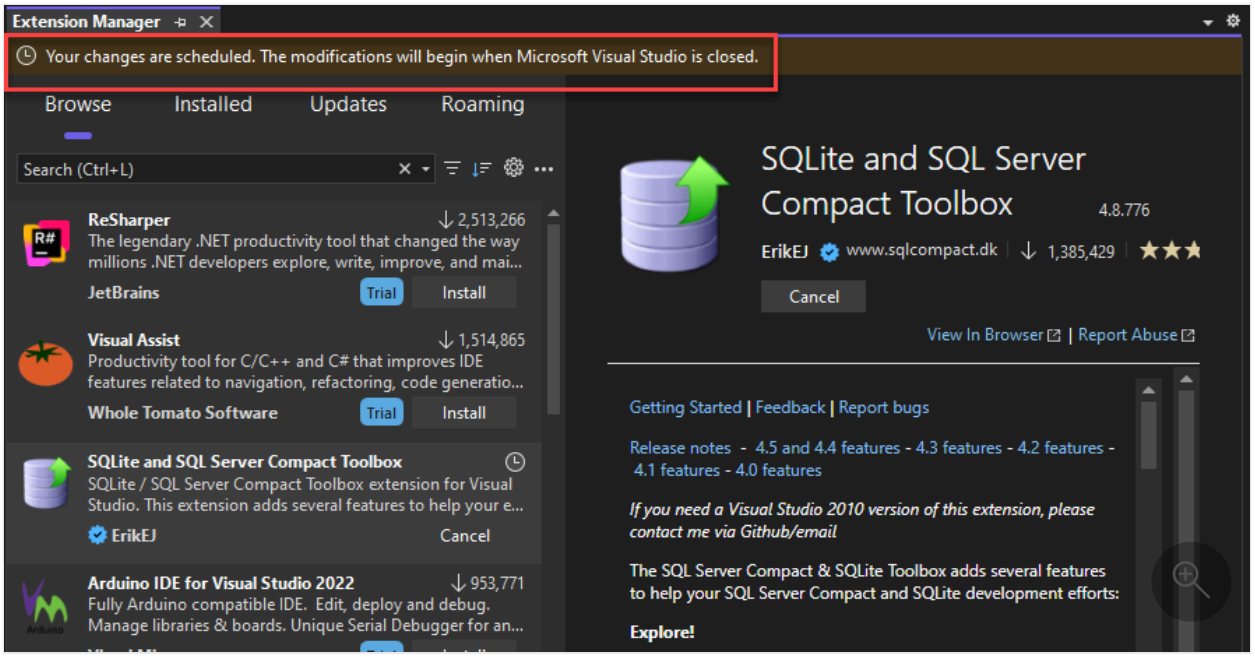
You can install extensions from Visual Studio Marketplace or from Extension Manager in Visual Studio.

To install extensions from Visual Studio:

1. Select **Extensions** > **Manage Extensions**. Find the extension you want to install. (If you know the name or part of the name of the extension, you can search in the search box.)

2. Select **Install**.

After the download completes, you see a notification at the top of Extension Manager: "Your changes are scheduled. The modifications will begin when Microsoft Visual Studio is closed."



If you try to install an extension that has dependencies, the installer determines whether they're already installed. If they aren't installed, Extension Manager lists the dependencies that must be installed before you can install the extension.

Install without using Extension Manager

Extensions that are packaged in *.vsix* files might be available in locations other than Visual Studio Marketplace. The **Extensions** > **Extension Manager** dialog can't detect these files, but you can install a *.vsix* file by double-clicking the file or selecting the file and then selecting **Enter**. If you run into permission issues, ensure you are running Visual Studio as an administrator. After that, just follow the instructions. When the extension is installed, you can use Extension Manager to enable it, disable it, or uninstall it.

> **Note**
>
> - Visual Studio Marketplace contains both .vsix-based and .msi-based extensions. Extension Manager can't enable or disable .msi-based extensions.
> - If an .msi-based extension includes an *extension.vsixmanifest* file, the extension appears in **Extension Manager**.

Uninstall or disable an extension

If you want to stop using an extension, you can either disable it or uninstall it. Disabling an extension keeps it installed but unloaded. Find the extension and select **Uninstall** or

Disable. Restart Visual Studio to unload a disabled extension.

> **Note**
>
> You can disable VSIX-based extensions but not extensions that are installed via MSI. MSI-installed extensions can only be uninstalled.

Manage extensions

This section provides information about managing extensions for an organization, automatic extension updates, and unresponsiveness notifications.

Per-user and administrative extensions

Most extensions are per-user extensions and are installed in the *%LocalAppData%\Microsoft\VisualStudio\<Visual Studio version>\Extensions* folder. A few extensions are administrative extensions and are installed in the *<Visual Studio installation folder>\Common7\IDE\Extensions* folder.

To protect your system against extensions that might contain errors or malicious code, you can restrict per-user extensions to load only when Visual Studio is run with normal user permissions. This causes per-user extensions to be disabled when Visual Studio is run with elevated permissions.

To restrict when per-user extensions load:

1. Open the extensions options page (**Tools** > **Options** > **Environment** > **Extensions**).
2. Clear the **Load per user extensions when running as administrator** checkbox.
3. Restart Visual Studio.

Automatic extension updates

Extensions are updated automatically when a new version is available on Visual Studio Marketplace. The new version of the extension is detected and installed in the background. The next time you open Visual Studio, the new version of the extension will be running.

If you want to disable automatic updates, you can disable the feature for all extensions or for only specific extensions.

- To disable automatic updates for all extensions, select the **View Settings** button in the **Extensions** > **Manage Extensions** dialog. In the **Options** dialog, under **Environment** > **Extensions**, clear **Automatically update extensions**.
- To disable automatic updates for a specific extension, clear the **Automatically update this extension** option in the extension's details pane on the right side of Extension Manager.

Crash and unresponsiveness notifications

Visual Studio notifies you if it suspects that an extension was involved in a crash during a previous session. When Visual Studio crashes, it stores the exception stack. The next time Visual Studio starts, it examines the stack, starting with the leaf and working towards the base. If Visual Studio determines that a frame belongs to a module that's part of an installed and enabled extension, it shows a notification.

Visual Studio also notifies you if it suspects an extension is causing the UI to be unresponsive.

When you see one of these notifications, you can ignore it or take one of the following actions:

- Select **Disable this extension**. Visual Studio disables the extension and lets you know whether you need to restart your system for the disable operation to take effect. You can re-enable the extension in the **Extensions** > **Manage Extensions** dialog.
- Select **Never show this message again**.
  - If the notification concerns a crash in a previous session, Visual Studio no longer shows a notification when a crash associated with the extension occurs. Visual Studio still shows notifications when unresponsiveness can be associated with the extension or for crashes or unresponsiveness that can be associated with other extensions.
  - If the notification concerns unresponsiveness, the IDE no longer shows a notification when the extension is associated with unresponsiveness. Visual Studio still shows crash-related notifications for the extension and crash-related and unresponsiveness-related notifications for other extensions.
- Select **Learn more**.
- Select the **X** at the end of the notification to dismiss the notification. A new notification will appear if the extension is associated with a crash or with UI unresponsiveness in the future.

> **Note**
>
> A UI unresponsiveness notification or crash notification means that one of the extension's modules was on the stack when the UI was unresponsive or when the crash occurred. It doesn't necessarily mean that the extension caused the problem. It's possible that the extension called code that's part of Visual Studio, which in turn resulted in unresponsive UI or a crash. However, the notification might still be useful if the extension that led to the UI unresponsiveness or crash isn't important to you. In this case, disabling the extension avoids the UI unresponsiveness or crash in the future.

Samples

When you install an online sample, the solution is stored in two locations:

- A working copy is stored in the location that you specified when you created the project.
- A separate master copy is stored on your computer.

You can use the **Extensions** > **Manage Extensions** dialog box to perform these samples-related tasks:

- List the master copies of samples that you have installed.
- Disable or uninstall the master copy of a sample.
- Install Sample Packs, which are collections of samples that relate to a technology or feature.
- Install individual online samples.
- View update notifications when source code changes are published for installed samples.
- Update the master copy of an installed sample when there's an update notification.

Related content

- Visual Studio SDK
- Visual Studio Extensibility
- Visual Studio Marketplace

Feedback

Was this page helpful? Yes No

Developer resources documentation

Use command line tools, REST APIS, and more to interface with Azure DevOps programmatically.

Azure DevOps Command Line Interface (CLI)

GET STARTED

Get Started with Azure DevOps CLI

Sign in with a Personal Access Token (PAT)

Configure policies

REFERENCE

Azure DevOps CLI quick reference

Repos and Test CLI

REFERENCE

Git commands

TFVC repository commands

Test case management commands

Node CLI for Azure DevOps

GET STARTED

Cross-platform CLI for Azure DevOps (tfx-cli)

Azure DevOps REST API Reference

GET STARTED

Get started

Build

Core

Git

Work item tracking

Azure DevOps Demo generator

GET STARTED

About the Demo generator

Get started