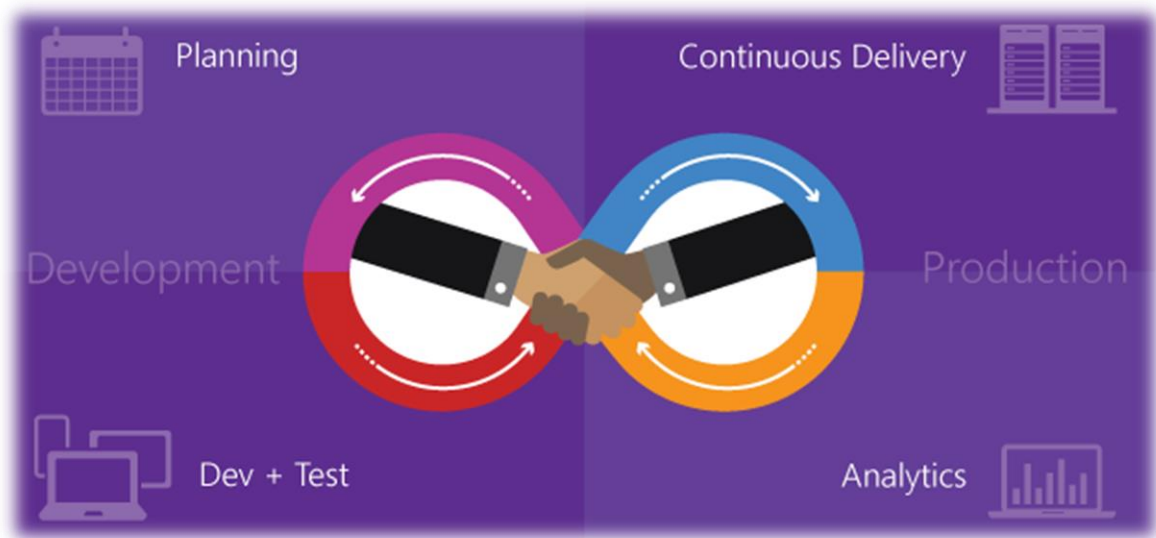


# Implementing DevOps with Team Foundation Server 2015



**Community Edition**

**By Suliman Al Battat**

# About the Author



**Suliman Al- Battat** is a senior consultant at KPMG with more than 10 years of intensive experience in the fields of ALM, Software Engineering, Software Development, Process Improvements, Operations, and others, along with a strong academic & professional skills. Providing ALM services for Software houses and enterprises cross MEA. Suliman is author for many books in ALM and Visual Studio Team Foundation Server (TFS), he is an authorized consultant and trainer and public speaker for Microsoft.

Al- Battat has a **Bachelor** degree in Software Engineering from Petra University in Jordan. Al-Battat also holds internationally-recognized awards in ALM and member of authorized worldwide ALM experts communities.

## Honors and Awards:

Microsoft Most Valuable Professional (MVP) – Visual Studio ALM, 2015	Microsoft
Microsoft Most Valuable Professional (MVP) – Visual Studio ALM, 2014	Microsoft
Microsoft Most Valuable Professional (MVP) – Visual Studio ALM, 2013	Microsoft
Microsoft Most Valuable Professional (MVP) – Visual Studio ALM, 2012	Microsoft
Microsoft Most Valuable Professional (MVP) – Visual Studio ALM, 2011	Microsoft
Visual Studio ALM Ranger, 2015	Microsoft
Visual Studio ALM Ranger, 2014	
Visual Studio ALM Ranger, 2013	Microsoft
Visual Studio ALM Ranger, 2012	Microsoft
Microsoft Most Active Expert And community contributor, 2010	Microsoft
Microsoft Student Partner, 2006	Microsoft
Microsoft.NET Champ, 2005	Microsoft

## Professional Affiliation:

- Member of Visual Studio ALM Rangers team.
- Member of Microsoft Jordan Dev tools professionals' team.
- Member of ALM worldwide communities and user groups
- Founder and leader of many ALM communities and user groups in MEA region.

# Glossary

ALM	Application Lifecycle Management
TFS	Team Foundation Server
DevOps	Development Operations
SP	SharePoint
AT	Application Tier
DT	Data Tier
AD	Active Directory
DC	Domain Controller
VS	Visual Studio
IDE	Integrated Development Environment
TE	Team Explorer
MSSCCI	Microsoft Source Code Control API
TFVC	Team Foundation Version Control

# Contents

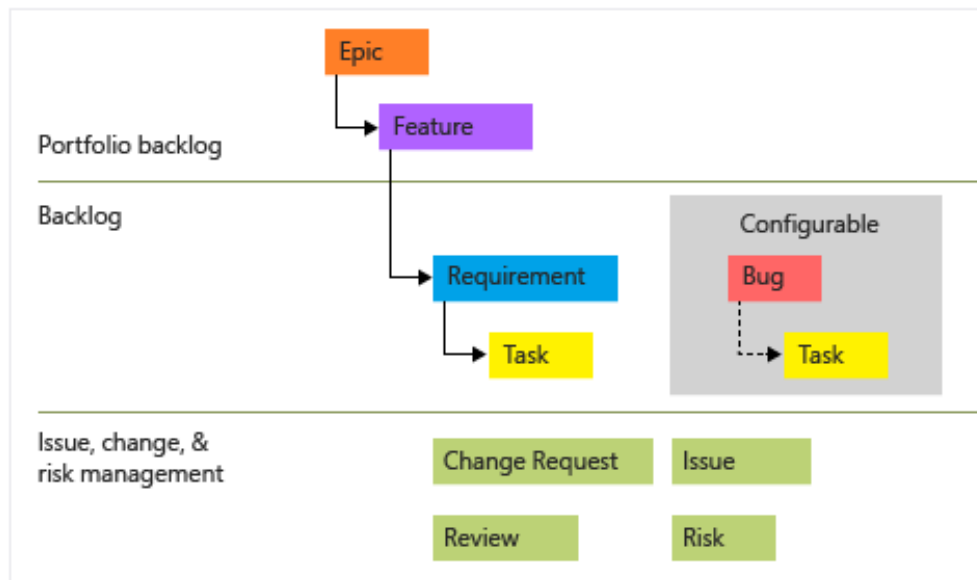
About the Author	2
Glossary	3
<b>1 Overview</b>	<b>6</b>
1.1 DevOps using CMMI process	6
<b>2 Requirement management</b>	<b>7</b>
2.1 Define requirements	7
2.2 Storyboard requirements and ideas using PowerPoint	10
2.3 Track requirements progress	17
<b>3 Implement development task (Task Management)</b>	<b>18</b>
3.1 Define task	18
3.2 Design documents	20
3.3 Design review	20
3.4 Verify quality	21
<b>3.5 Create a review work item for the design</b>	<b>21</b>
3.6 Unite tests	21
3.7 Code analysis	21
3.8 Code review process	21
3.9 Refactor code	22
3.10 Integrate changes	23
<b>4 Test management</b>	<b>24</b>
4.1 Define manual test cases	24
4.2 Running manual tests	26
4.3 Track your test results	29
<b>5 Bugs reporting and tracking</b>	<b>37</b>

5.1	Capture bugs	37
5.2	Track bugs as requirements or tasks	39
<b>6</b>	<b>Build and release management</b>	<b>43</b>
6.1	Understanding Microsoft release management	43
6.2	Release management workflow	43
<b>7</b>	<b>Reporting</b>	<b>108</b>
7.1	Dashboards	108
7.2	Manage dashboards	111

# 1 Overview

## 1.1 DevOps using CMMI process

Teams use the work item types (WITs) provided with the MSF for CMMI Process Improvement 2015 (CMMI) process template to plan and track progress of software projects. Teams define requirements to manage the backlog of work and then, using the Kanban board, track progress by updating the status of requirements.



To gain insight into a portfolio of requirements, Business analysts can map requirements to features. When teams work in iterations, they define tasks that automatically link to requirements.

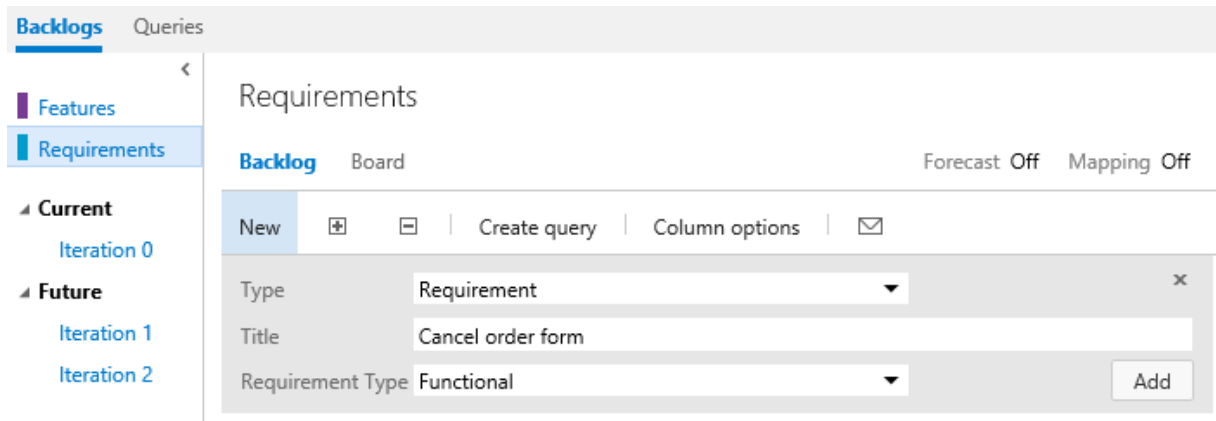
Using Microsoft Test Manager and the web portal, testers create and run test cases and define bugs to track code defects.

To support additional CMMI processes, teams can track change requests, risks, issues, and notes captured in review meetings.

## 2 Requirement management

### 2.1 Define requirements

You can create requirements from the quick add panel on the product backlog page using the web portal for TFS. Or using the team explorer client over visual studio as the following:

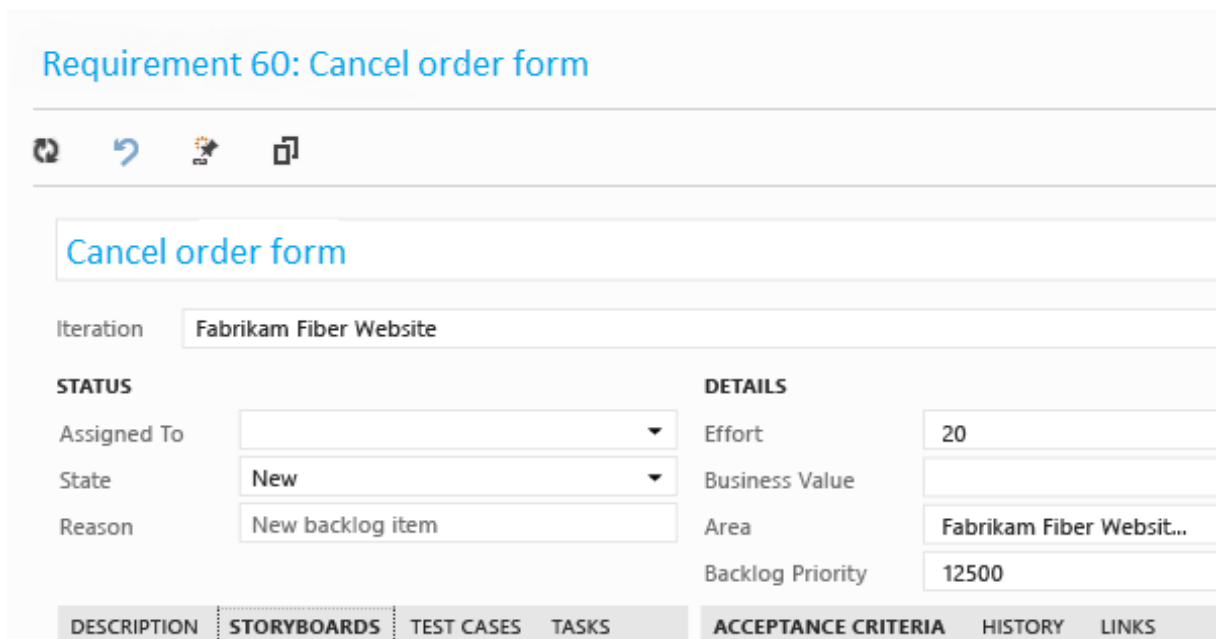


The screenshot shows the 'Requirements' backlog in TFS. On the left, a sidebar lists 'Features' and 'Requirements' (selected). Under 'Requirements', there are sections for 'Current' (Iteration 0) and 'Future' (Iteration 1, Iteration 2). The main area is titled 'Requirements' and has tabs for 'Backlog' (selected) and 'Board'. It includes 'Forecast Off' and 'Mapping Off' options. A 'New' button is visible. Below it, a form for creating a new requirement is shown with the following fields:

Type	Requirement
Title	Cancel order form
Requirement Type	Functional

An 'Add' button is located at the bottom right of the form.

Later, you can open each requirement to provide more details and estimate its size.



The screenshot shows the details page for 'Requirement 60: Cancel order form'. The title is 'Requirement 60: Cancel order form'. Below the title, there are icons for refresh, undo, redo, and a link icon. The main content area is titled 'Cancel order form'. Below this, there is a section for 'Iteration' with the value 'Fabrikam Fiber Website'. The 'STATUS' section includes 'Assigned To' (empty), 'State' (New), and 'Reason' (New backlog item). The 'DETAILS' section includes 'Effort' (20), 'Business Value' (empty), 'Area' (Fabrikam Fiber Websit...), and 'Backlog Priority' (12500). At the bottom, there are tabs for 'DESCRIPTION', 'STORYBOARDS', 'TEST CASES', 'TASKS', 'ACCEPTANCE CRITERIA', 'HISTORY', and 'LINKS'.

Requirements specify the functions and product elements that teams need to create. Business analysts typically define and stack rank requirements on the product backlog page. The team then scopes the size of the effort to deliver the highest priority items.

Use the following guidance and that provided for fields used in common across work item types when filling out the form:

Field	Usage
Description	<p>Provide enough detail for estimating how much work will be required to implement the requirement. Focus on who the requirement is for, what users want to accomplish, and why. Don't describe how the requirement should be developed. Do provide sufficient details so that your team can write tasks and test cases to implement the item.</p> <p>In HTML fields, you can add rich text and images.</p>
Impact Assessment	<p>The customer impact of not implementing this requirement. You might include details from the Kano model about whether this requirement is in the surprise, required, or obvious categories. You capture this information in the rich-text HTML field which corresponds to Impact Assessment.</p>
(Requirement) Type (Required)	<p>The kind of requirement to implement. You can specify one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>Business Objective</b></li> <li>• <b>Feature</b> (default)</li> <li>• <b>Functional</b></li> <li>• <b>Interface</b></li> <li>• <b>Operational</b></li> <li>• <b>Quality of Service</b></li> <li>• <b>Safety</b></li> <li>• <b>Scenario</b></li> <li>• <b>Security</b></li> </ul>
Value area	<p>The area of customer value addressed by the epic, feature, requirement, or backlog item. Values include:</p> <ul style="list-style-type: none"> <li>• <b>Architectural</b> : Technical services to implement business features that deliver solution</li> <li>• <b>Business</b>: Services that fulfill customers or stakeholder needs that directly deliver customer value to support the business (Default)</li> </ul>
Size	<p>Estimate the amount of work required to complete a requirement using any unit of measurement your team prefers, such as t-shirt size, story points, or time.</p> <p>By defining the <b>Size</b> for requirements, teams can use the Agile velocity charts and forecast tools to estimate future iterations or work efforts. The Kanban Cumulative</p>



	Flow Diagram references the values in this field. For additional guidance, see the Estimating white paper.
Original Estimate	<p>The amount of estimated work required to complete a task. Typically, this field doesn't change after it is assigned.</p> <p>You can specify work in hours or in days. There are no inherent time units associated with this field.</p>
Start Date/Finish Date	<p>The target dates for when the work will start or finish. These fields are filled in by Microsoft Project when you use it for scheduling.</p> <p>You can specify work in hours or in days. There are no inherent time units associated with this field.</p>
Priority (Required)	<p>A subjective rating of the requirement as it relates to the business. Allowed values are:</p> <ul style="list-style-type: none"> <li>• <b>1:</b> Product cannot ship without the item.</li> <li>• <b>2:</b> (default) Product cannot ship without the item, but it doesn't have to be addressed immediately.</li> <li>• <b>3:</b> Implementation of the item is optional based on resources, time, and risk.</li> </ul>
Triage (Required)	<p>Indicates the type of triage decision that is pending for the work item. Use this field when the work item is in the Proposed state and specify one of the following values: <b>Pending</b> (default), <b>More Info</b>, <b>Info Received</b>, and <b>Triaged</b>.</p>
Blocked	<p>Indicates whether a team member is prevented from making progress toward implementing a requirement or task or resolving a bug, change request, or risk. If an issue has been opened to track a blocking problem, you can create a link to the issue. You can specify <b>Yes</b> or <b>No</b>.</p>
Committed (Required)	<p>Indicates whether the requirement is committed in the project or not. You can specify <b>Yes</b> or <b>No</b> (default).</p>
Integrated In	<p>Product build number that incorporates the requirement, change request, or fixes a bug.</p>
User Acceptance Test (Required)	<p>The status of the user acceptance test for a requirement. You can specify one of the following values:</p> <ul style="list-style-type: none"> <li>• <b>Pass</b></li> <li>• <b>Fail</b></li> <li>• <b>Not Ready</b> (default)</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>Ready</b></li> <li>• <b>Skipped</b></li> <li>• <b>Info Received</b></li> </ul> <p>You specify Not Ready when the requirement is in the Active state, and you specify Ready when the requirement is in the Resolved state.</p>
Subject Matter Experts	The names of team members who are familiar with the customer area that this requirement represents.



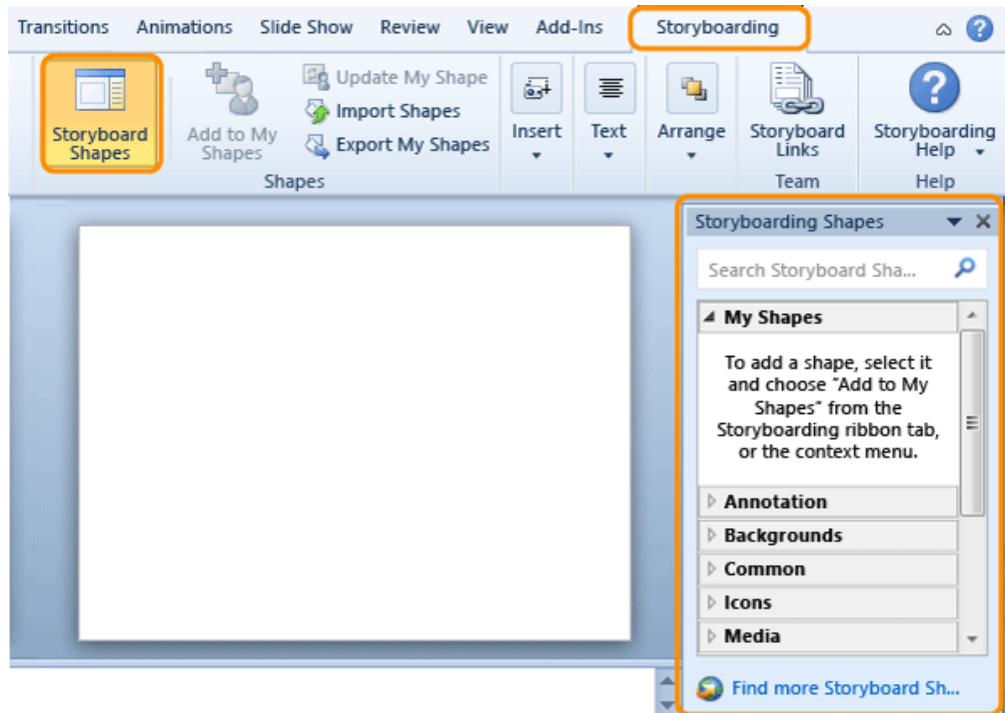
Use the **History** section to add and review comments made about the work being performed.

## 2.2 Storyboard requirements and ideas using PowerPoint

With storyboarding, you turn the requirements, ideas and goals into something visual. Your ideas are easier for other people to understand, so they can give you constructive feedback, sooner. You can bring your ideas to life with storyboard shapes, text, animation, and all the other features that PowerPoint Storyboarding provides.

### 2.2.1 Use storyboard shapes and PowerPoint features

1. Open Power Point Storyboarding and start with a blank slide. You should see the **Storyboarding** ribbon and **Storyboard Shapes** library.



Or, you can open PowerPoint Storyboarding from the **Storyboarding** tab of a backlog work item (Visual Studio or the web portal for TFS).

**Requirement 60: Cancel order form**

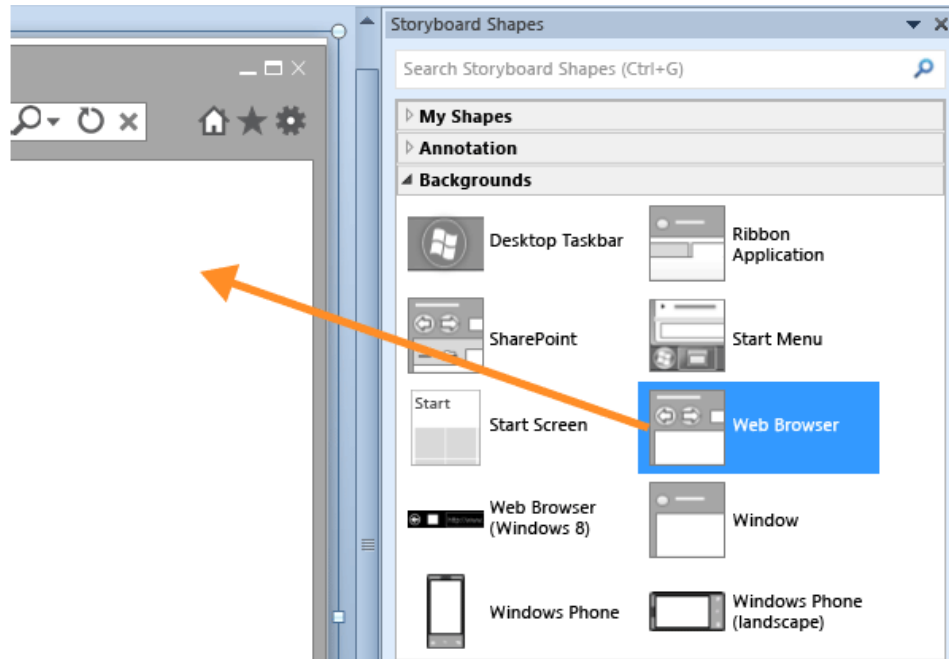
Cancel order form

Iteration: Fabrikam Fiber Website

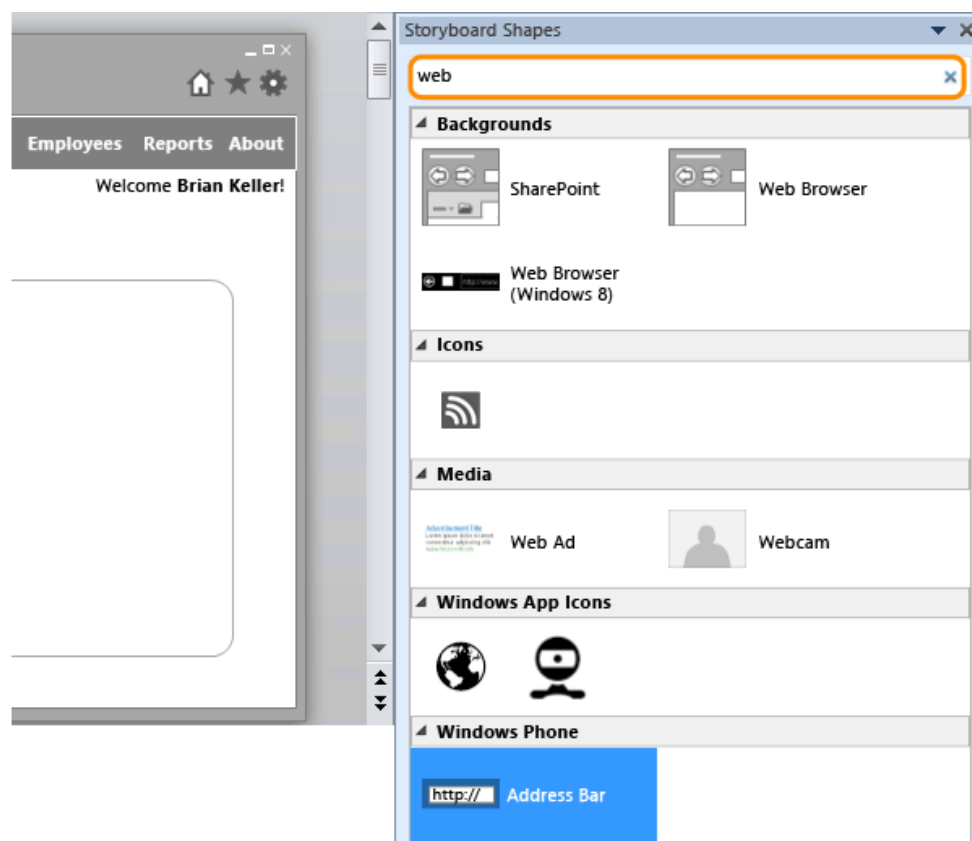
STATUS		DETAILS	
Assigned To		Effort	20
State	New	Business Value	
Reason	New backlog item	Area	Fabrikam Fiber Websit...
		Backlog Priority	12500

DESCRIPTION	STORYBOARDS	TEST CASES	TASKS	ACCEPTANCE CRITERIA	HISTORY	LINKS
<div>Start storyboarding</div> <div>Title</div>						

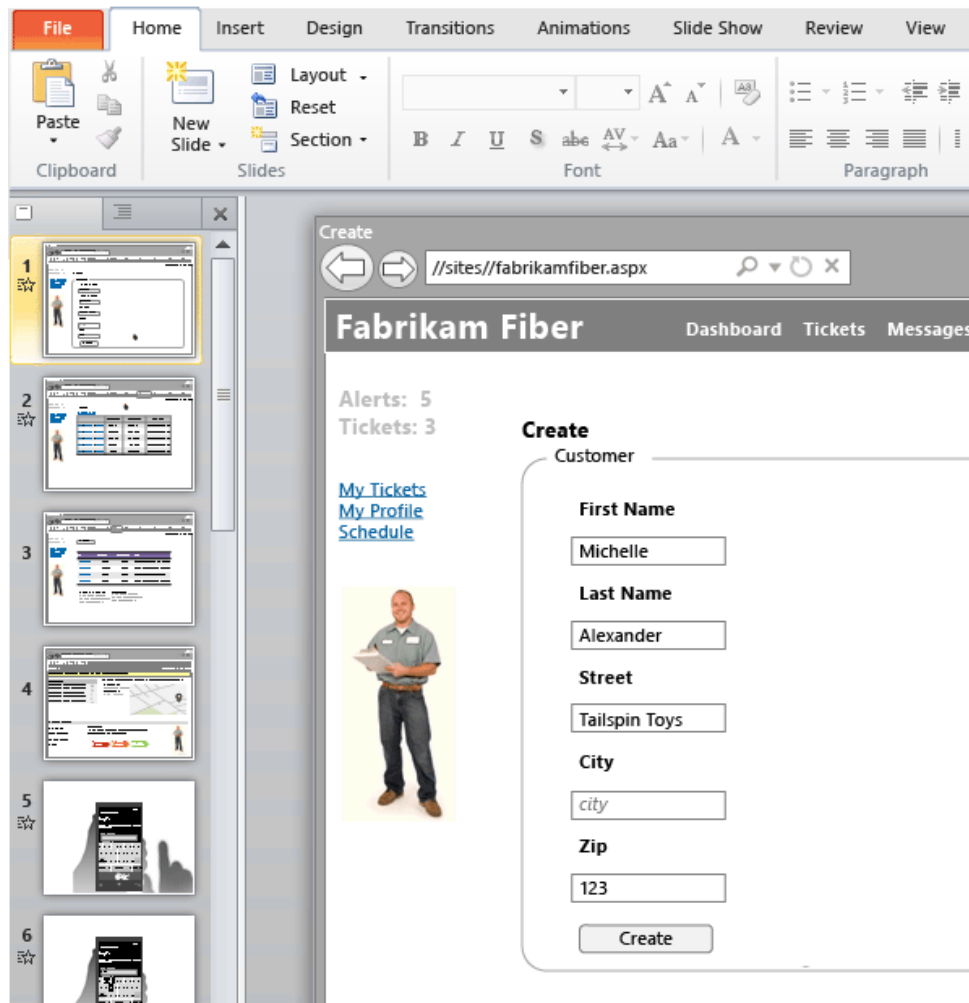
2. Add a background shape that's appropriate for your app. To add a shape, just drag it onto the slide.



3. Search for more shapes to complete your design.



4. Create more slides to show the flow of your app. Share them with your team to get early feedback on the design.



## Tip!

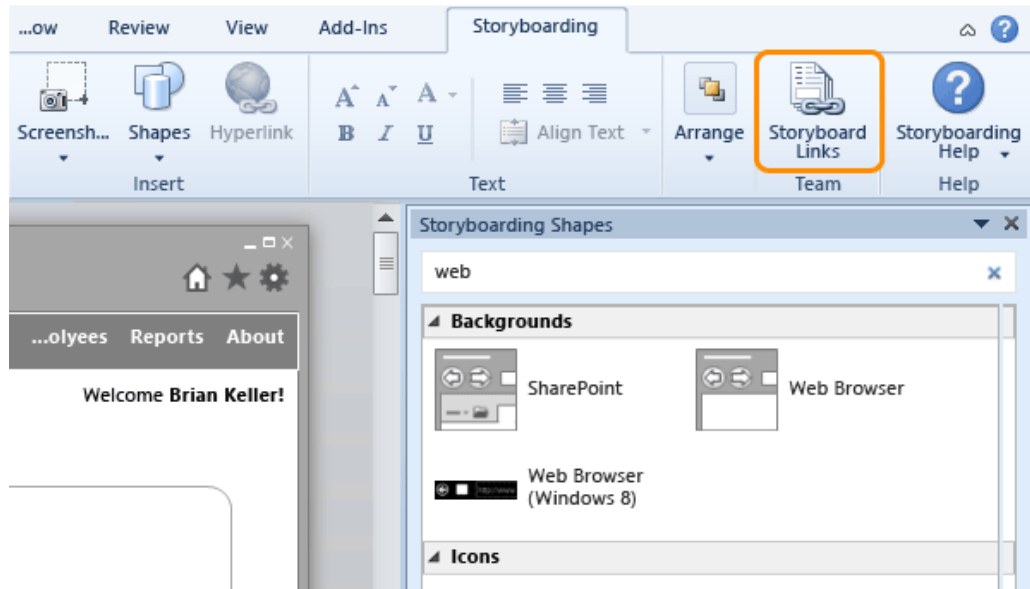
- Use animation to bring your flow to life.
- Take screenshots of your apps. For example, add a screenshot as the background of a master slide.
- Use [Office MIX](#) to create a demo of your feature.

## 2.2.2 Link storyboard to a backlog item

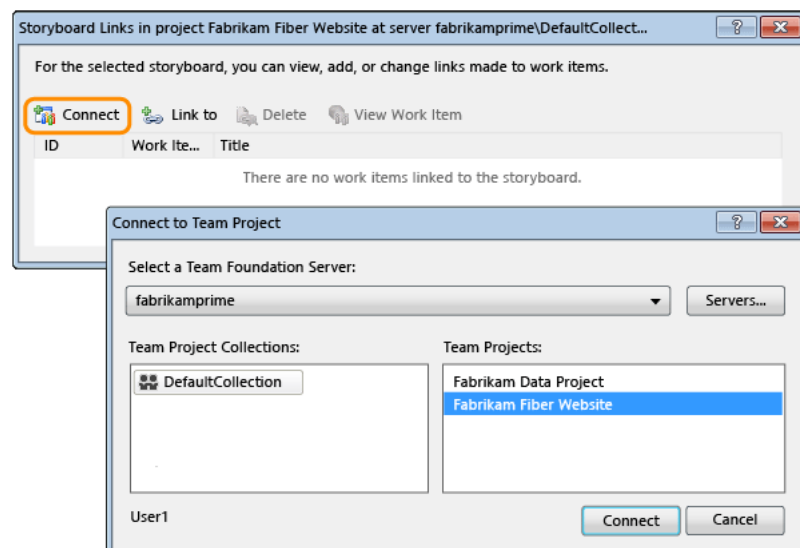
When you share your storyboards to a shared network location, you can link the storyboards to a backlog item. That way, your team members will be able to open the storyboards from the work items and annotate them with their suggestions.

1. Save or upload your storyboard to a shared location that everyone on your team can access.
2. If you started Power Point Storyboarding from a backlog item, then you are already linked to the initial item and you are done.

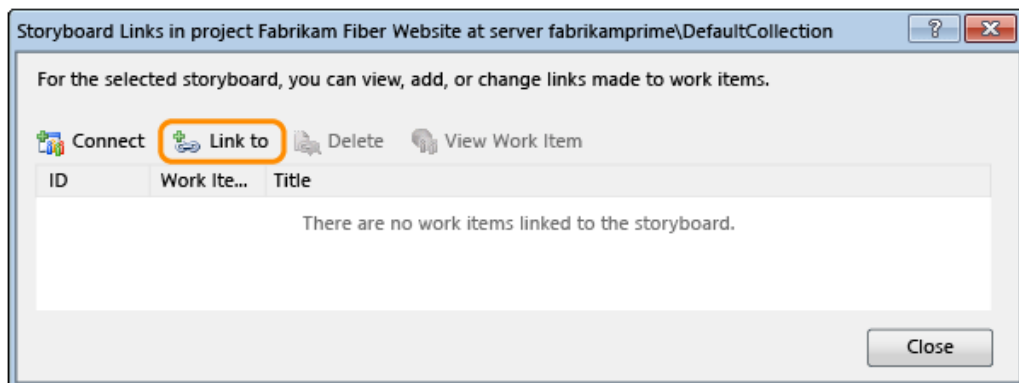
Otherwise, open **Storyboard Links**.



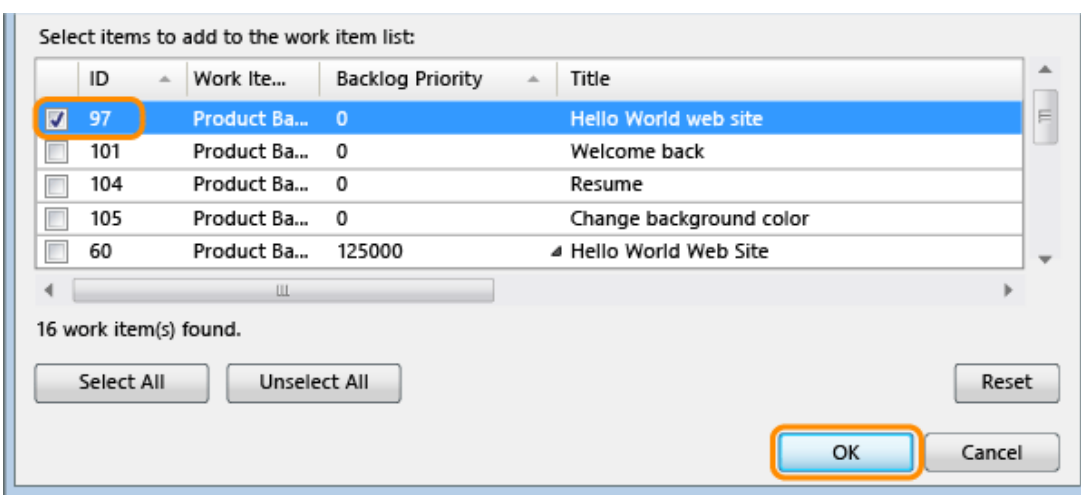
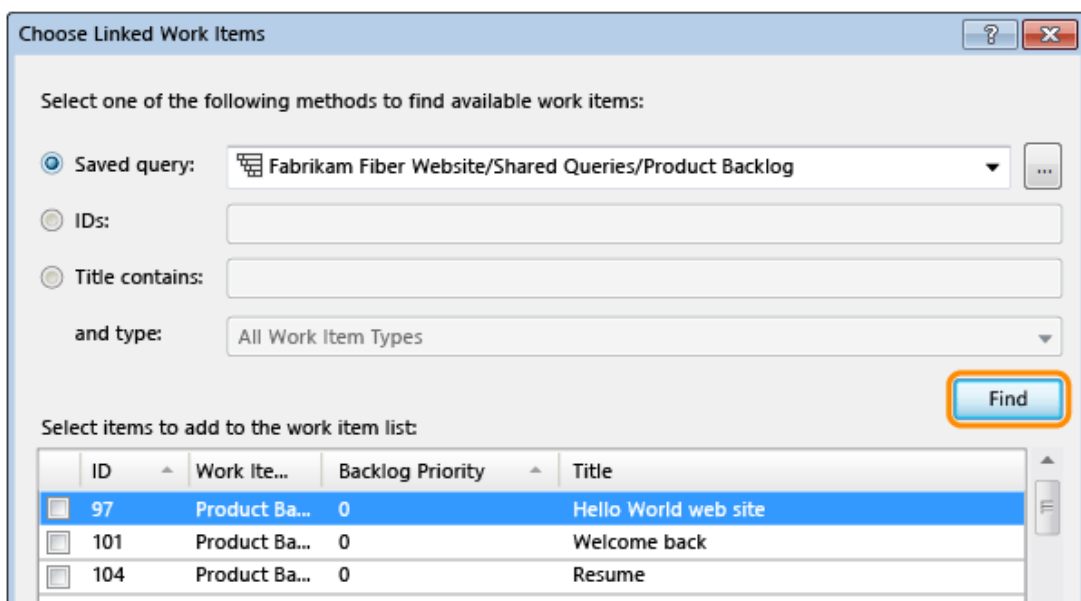
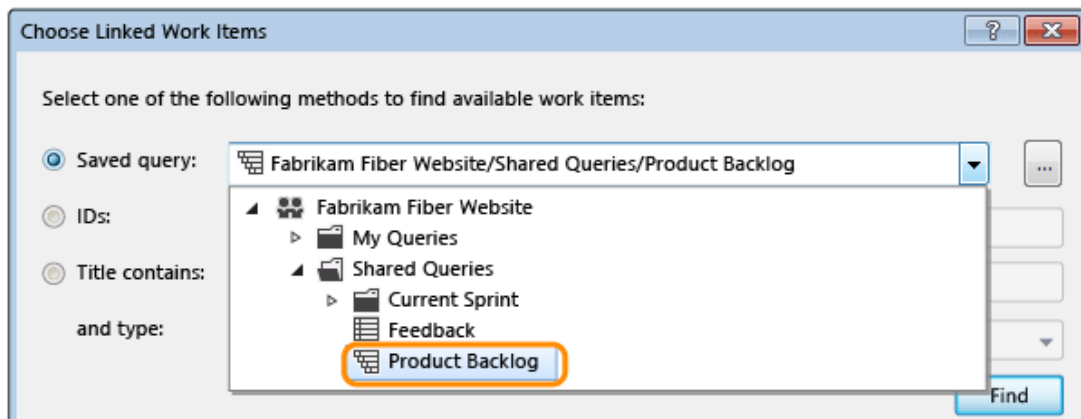
3. Connect to a team project.



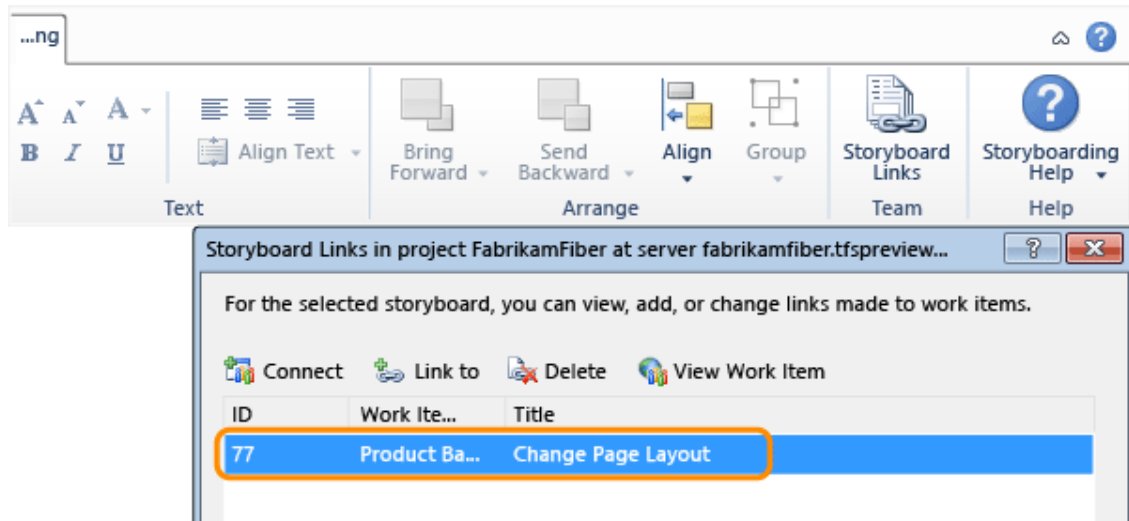
4. Then, link to a work item.



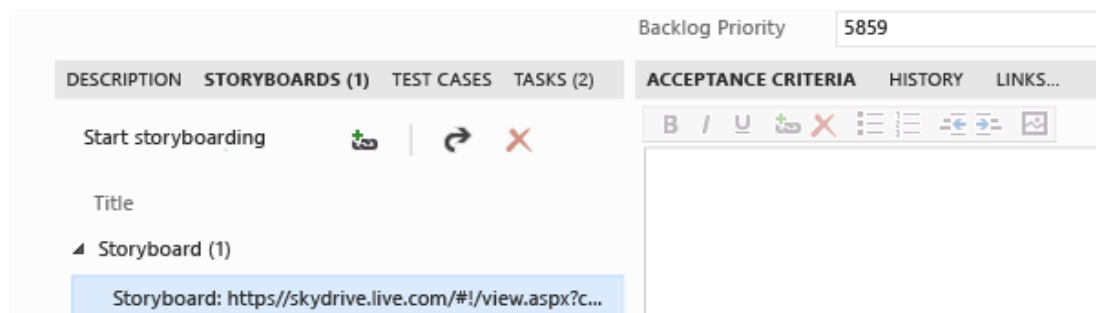
5. Select a work item to link to. The next screenshot shows how to do that using a saved query. You can also do a simple search on the title, or just provide the ID.



6. Now the storyboard is linked to the work item.



7. And, whoever views the work item can also access the storyboard.



8. With PowerPoint Storyboarding, you can illustrate a new or a modified interface. You can capture existing user interfaces and build a storyboard from a collection of predefined storyboard shapes. Also, you can customize the slide layouts for your web, client, or phone applications. And, by linking the storyboard to the product backlog item or user story, you automatically share it with your team

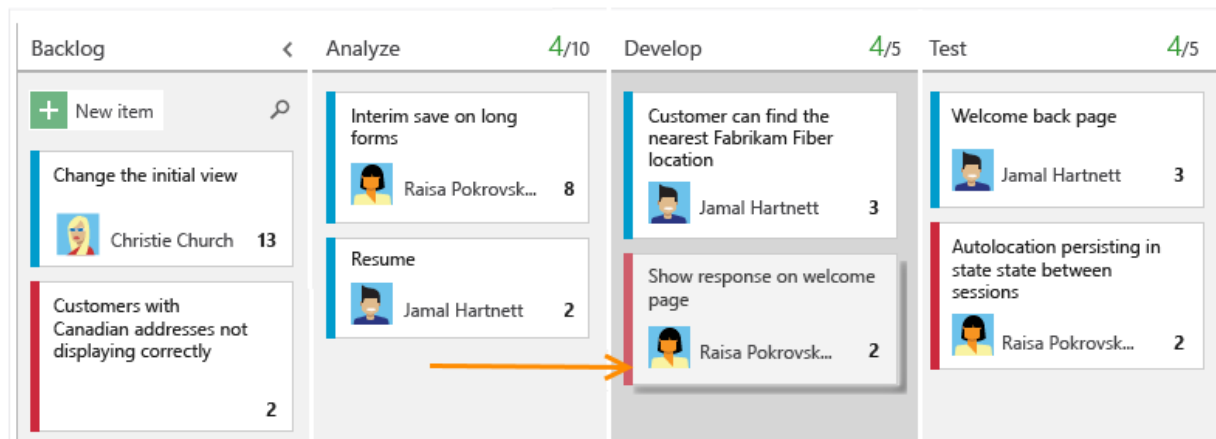


- The only way to get the TFS Storyboarding add-in is by installing one of the latest editions of Visual Studio or [Team Foundation Server Standalone Office Integration 2015 Update 1](#).
- You can create your own shapes using My Shapes, which get saved to the following folder in a .sbsx file:  
  
\*Drive\*:\Users\\*UserName\*\AppData\Local\Microsoft\Team Foundation\6.0\PowerPoint\Shapes



## 2.3 Track requirements progress

Teams can use the **Kanban board** to track progress of requirements, and the **Iteration task board** to track progress of tasks. Dragging items to a new state column updates the workflow **State** and **Reason** fields as the following:



You can customize the Kanban board to support additional swim lanes or columns.

A typical workflow progression for a requirement follows:


- The product owner creates a requirement in the **Proposed** state with the default reason, **New requirement**.
- The product owner updates the status to **Active** when they begin work to implement it.
- The team updates the status to **Resolved** when code development is finished and system tests have passed.
- Lastly, the team or product owner moves the requirement to **Closed** when the product owner agrees that it has been implemented according to the Acceptance Criteria and passed all validation tests.

## 3 Implement development task (Task Management)

### 3.1 Define task

When your team manages their work in iterations, they can use the iteration backlog page to break down the work to be accomplished into distinct tasks.

A development task is a small piece of development work that stems from a requirement. Implementing a development task involves adding the appropriate new functionality to your software. After you complete a development task, it should be unit tested, reviewed, code analyzed, and integrated into the existing code base.

Fabrikam Fiber Website Team Iteration 0				August 26 - September 23 21 work days remaining
<b>Backlog</b> Board Capacity				Work details
+ - Create Query Column Options				
Title		State	Assigned To	Remaining Work
	Hello World Web Site	Proposed		
	Add an information form	Proposed		

Name the task and estimate the work it will take. Estimating the cost of development tasks helps control the scope of features and schedule development work. Cost estimates for all development tasks should be completed and any issues should be resolved before the iteration planning meeting. If the total cost of the development tasks is more than can be done in an iteration, a task must be deferred or reassigned. After a development task is chosen, it is the responsibility of the developer to cost the task.

Create a task work item for each development task that is chosen, and link it to the requirement from which it was created. This is accomplished from the Implementation tab on either the task or the requirement work item. Base your estimates on the time that was required to complete similar tasks, and be sure to factor in the cost of writing unit tests. For each task, enter the estimate into the Original Estimate field of the task work item.

The form for task work items stores data in the fields and tabs that the following illustrations show:

New Task 1\*: Welcome Screen

Copy template URL

Tags
Add...

Welcome Screen

STATUS

Assigned To
State
Reason

PLANNING

Priority
Triage
Blocked

CLASSIFICATION

Area
Iteration
Task Type
Discipline

EFFORT (HOURS)

Original Estimate
Remaining work
Completed work

DESCRIPTION

IMPLEMENTATION (1)

OTHER

HISTORY

ATTACHMENTS

ALL LINKS (1)



Estimate the tasks by minimum 1 day and maximum 3 days as a best practice to manage and track the work progress with team members.

After tasks have been created and estimated, use the Work Breakdown query to view the breakdown of all your requirements and tasks.

The following table describes all task fields:

Field	Usage
Task Type	<p>Select the kind of task to implement from the allowed values:</p> <ul style="list-style-type: none"> <li>Corrective Action</li> <li>Mitigation Action</li> <li>Planned</li> </ul>
Discipline	<p>Select the discipline this task represents when your team estimates sprint capacity by activity.</p> <ul style="list-style-type: none"> <li>Analysis</li> <li>Development</li> <li>Test</li> <li>User Education</li> <li>User Experience</li> </ul> <p>This field is also used to calculate capacity by discipline. It is assigned to <code>type="Activity"</code> in the ProcessConfiguration file.</p>

Field	Usage
Original Estimate	The amount of estimated work required to complete a task. Typically, this field doesn't change after it is assigned.
Remaining Work	<p>Indicate how many hours or days of work remain to complete a task. As work progresses, update this field. It's used to calculate capacity charts, the sprint burndown chart, and the Burndown and Burn Rate Report.</p> <p>If you divide a task into subtasks, specify hours for the subtasks only. You can specify work in any unit of measurement your team chooses.</p> <p>The amount of work remaining to complete a task. As work progresses, update this field. It's used to calculate capacity charts, the sprint burndown chart, and the Sprint Burndown report.</p> <p>If you divide a task into subtasks, specify hours for the subtasks only. You can specify work in any unit of measurement your team chooses.</p>
Completed Work	The amount of work that has been spent implementing a task.

## 3.2 Design documents

Your design documents should include enough information to describe to a developer how to write code to implement the requirement in the product.

Design documents can be a collection of specifications, requirement work items, and other documents, depending on your team process.

Consider using design patterns, object-oriented design, structural models, modeling languages, entity relationship models, and other techniques in the guidelines for the design that is determined for your team. It is also a good idea to document the rationale for key decisions that were made. For example, if there is a significant effect on cost, schedule, or technical performance, document the reason for the decisions behind these effects, and include that information in your design.

After you create the necessary design documents, store them in the document library where your team members can access them.

## 3.3 Design review

A design review is used to ensure that the new or revised design is technically accurate, complete, testable, and of high quality and that it implements the requirement correctly. Design reviews are a key method of guaranteeing quality early by identifying problems before they appear in the code. Design reviews also provide additional insight about the design from other developers.

The developer who is responsible for creating the design should organize the design review by identifying reviewers, scheduling the review, and distributing the design to all the reviewers.

Any stakeholders who are involved or affected by the design should participate in the review. Typically this might include the project manager, the lead developer, and the **tester for the design area**. All developers who are on the same team as the developer whose code is being reviewed should also participate in the review.

Schedule the review meeting, and distribute the design documents early enough to give each reviewer sufficient time to read them. Plan the length of the review meeting to correspond to how many technical details must be reviewed.

### 3.4 Verify quality

Ensure that the design is testable. Does it build code that cannot be verified or validated in a reasonable manner? If so, you cannot ensure the quality of the code, and the design must be reworked. Examine the design documents for problems that will lead to code errors. Look for incorrect interface descriptions, design mistakes, or naming confusion. Compare the design documents against existing criteria, such as operator interface standards, safety standards, production constraints, design tolerances, or parts standards. Create bug work items that describe any flaws that are found in the design documents, and assign them to the responsible developer.

### 3.5 Create a review work item for the design

A review work item is created to document the results of the design review. The review team must decide the next steps for the design, which depend on the magnitude of the changes necessary. If no changes are necessary, set the State of the work item to Closed, set the Reason to Accepted (as is), and note that coding can start on the design. If minor changes are necessary, set the State of the work item to Resolved, and set the Reason to Accepted with Minor Changes. This indicates that coding can start after the minor changes have been implemented in the design. If major changes are necessary, set the State of the work item to Resolved, and set the Reason to Accepted with Major Changes. The design must be reworked and another design review must be performed before coding can start on the design.

### 3.6 Unite tests

Unit tests verify the correct implementation of a unit of code. Writing and performing unit tests identifies bugs before testing starts and, therefore, helps reduce the cost of quality control. Developers must write unit tests for all code that will be written as part of implementing a development task or fixing a bug.

### 3.7 Code analysis

Code analysis checks code against a set of rules that help enforce development guidelines. The goal of code analysis is to have no code analysis violations or warnings. Code analysis can inspect your code for more than 200 potential issues in naming conventions, library design, localization, security, and performance.

If you start to run code analysis early in your development cycle, you can minimize violations and warnings on an ongoing basis.

However, if you run code analysis on existing code that has not been checked before, you may have many rule violations. If this is the case, you might want to create a baseline set of critical rules that the code must pass and then expand the rule set as the more critical issues are resolved. That way, a team can move forward on new functionality as it improves its existing code base.

### 3.8 Code review process

The lead developer should organize the code review by identifying the reviewers, scheduling the code review, and sending the code for review to all reviewers. To prepare for the code review, take the following steps:

1. Create a review work item to track the decisions that are made in the review. If no changes are necessary, set the State of the work item to Closed, set the Reason to Accepted (as is), and note that coding can start on the design. If minor changes are necessary, set the State of the work item to

Resolved, and set the Reason to Accepted with Minor Changes, which indicates that coding can start after the minor changes have been implemented. If major changes are necessary, set the State of the work item to Resolved, and set the Reason to Accepted with Major Changes. The design must be reworked and another design review must be performed before coding can start on the design.

2. Determine who will participate in the code review. Typically, at least the lead developer and the architect who is responsible for the code area should participate in the review.
3. Schedule a review meeting with the reviewers, and allow sufficient time for each reviewer to read and understand the code before the meeting. Plan the length of the review meeting to correspond to how much code must be reviewed.

### 3.8.1 Code review

A code review is used to ensure that new or changed code meets an established quality bar before it is integrated into the daily build. Quality considerations are coding standards, conformance to architecture and design, performance, readability, and security. Code reviews also provide additional insight from other developers about how code should be written.

Verify Code Relevance	The code that is being reviewed is relevant to the task for which the code is written. No code changes should be allowed that do not address the functionality that is implemented or corrected.
Verify Extensibility	The code is written so that it can be extended, if that is the intent, or reused in other areas of the system.  String constants that are used in the code are correctly put in resources that can be internationalized.
Verify Minimal Code Complexity	Repeated code can be simplified into common functions.  Similar functionality is put in a common procedure or function.
Verify Algorithmic Complexity	The number of execution paths in the code that is reviewed is minimized.
Verify Code Security	The code is checked for the protection of assets, privilege levels, and the use of data at entry points.

## 3.9 Refactor code

Code is refactored after a code review has determined that changes must be made to address code quality, performance, or architecture.

Read the code review work item notes to determine how you will refactor the code.

Apply the refactoring incrementally, one change at a time. Change the code and all references to the modified area as necessary.

Perform unit tests so that the area remains semantically equivalent after the refactoring. Fix any unit tests that do not work. Perform code analysis, and fix any warnings. Perform unit tests again if code changes are made as a result of code analysis.

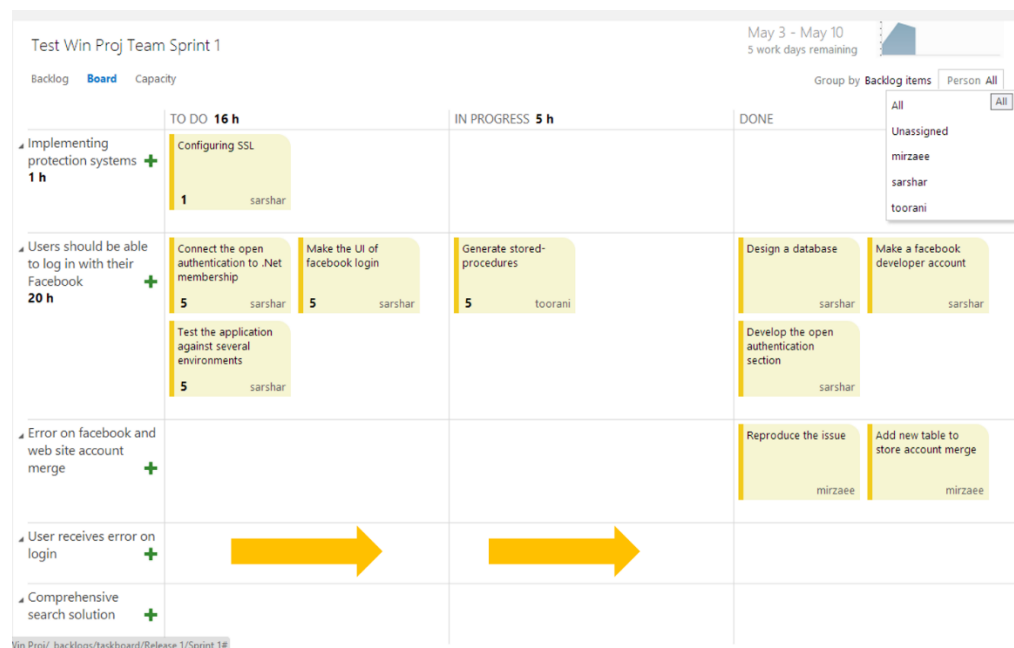
### 3.10 Integrate changes

The final step is to integrate the changes by checking them in to version control. Before code is checked in, any tests that are required by your process should be performed.

If the work item that is associated with the changes is a scenario or a quality of service requirement of which you are not the owner, notify the owner that the changes are complete. Set the task work item to Resolved, and assign it to one of the testers who created the test cases for the work item.

If the work item that is associated with the changes is a bug, set the bug work item to Resolved, and assign it to the original person who created it.


Leads can track development tasks with the developers and change the states using the task board



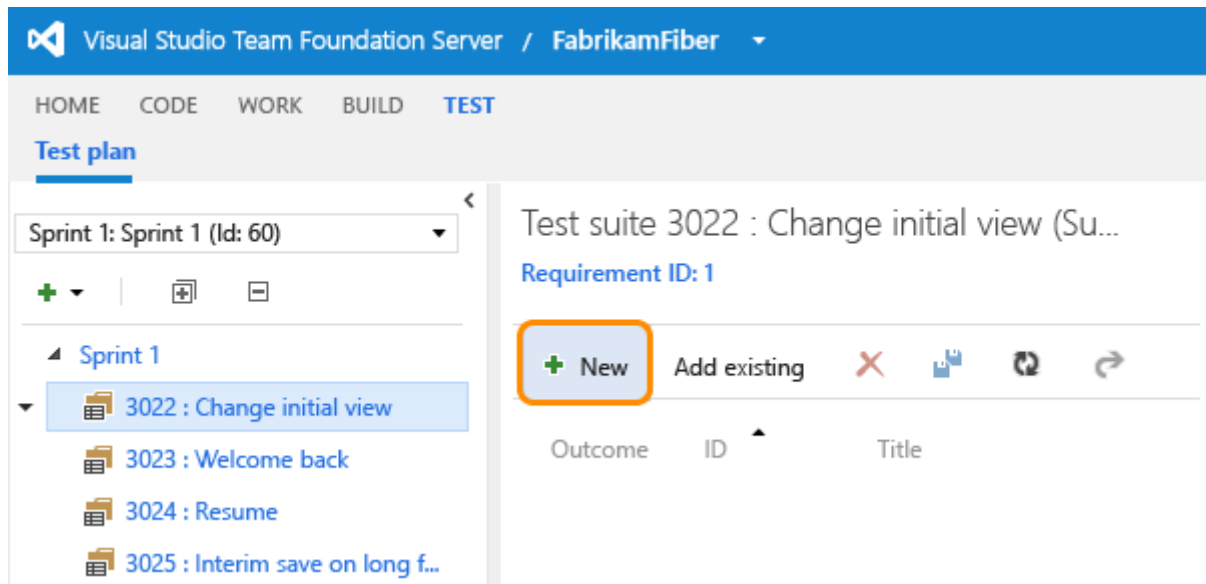
## 4 Test management

### 4.1 Define manual test cases

Create manual test cases to check that each of the deliverables meet your users' needs and requirements. You organize your test cases by adding test cases to test suites.

From the web portal or Test Manager, you can create test cases that automatically link to a requirement or bug. Or, you can link a requirement to a test case from the  (links tab).

1. From the Test hub in Team Web Access, choose a requirement-based test suite. Then create a test case for that suite.



A requirement-based test suite is created from a backlog item. If you add a test case to this type of suite, it is automatically linked to that backlog item.

2. Add test steps with actions and expected results so that any team member can run the test. You can add attachments to a step if you want.



Test Cases 3717: Change colors on initial view

Tags [Add...](#)

Change colors on initial view

STATUS		CLASSIFICATION	
Assigned To	Jamal Hartnett	Area	FabrikamFiber
State	Design	Iteration	FabrikamFiber\Release1\Spri...
Priority	2		
Automation status	Not Automated		

STEPS SUMMARY TESTED USER STORIES (1) ALL LINKS (2) ATTACHMENTS (1) ASSOCIATED AUTOMAT...

Click to type here to add a step

Action	Expected Result	Attachments
1. Open the home page for the web site	Home page is displayed	
2. Click settings icon	Settings page is displayed	
3. Change the default template to modern and click submit	The home page is displayed with the modern look see attached screenshot	<a href="#">homepagemodern.png (49K)</a>

3. Now you have created a test case that you can run.

The test case contains a number of fields, many of which are automated and integrated with Test Manager and the build process. The following table describes the fields that are defined in the test work item:

Field name	Description
Automation Status	The status of a test case. You can specify the following values: <ul style="list-style-type: none"> <li>Not Automated</li> <li>Planned</li> </ul>
Found In	Product build number, also known as a revision, in which a bug was found.
Integration Build	Product build number that incorporates the code or fixes a bug.
Issue	Indicates that the Shared Steps is associated with an expected result. Allowed values are Yes and No. (Shared Steps only)
Parameters	Contains the parameters to use when running a manual test. (Shared Parameters, Shared Steps, and Test Case)
Steps	The action and validation steps that are required to perform the test.
System Info	Information about the software and system configuration that is relevant to the test.
Steps to Reproduce	The steps that are required to reproduce unexpected behavior.
Test Suite Type	The test suite category. Allowed values are:

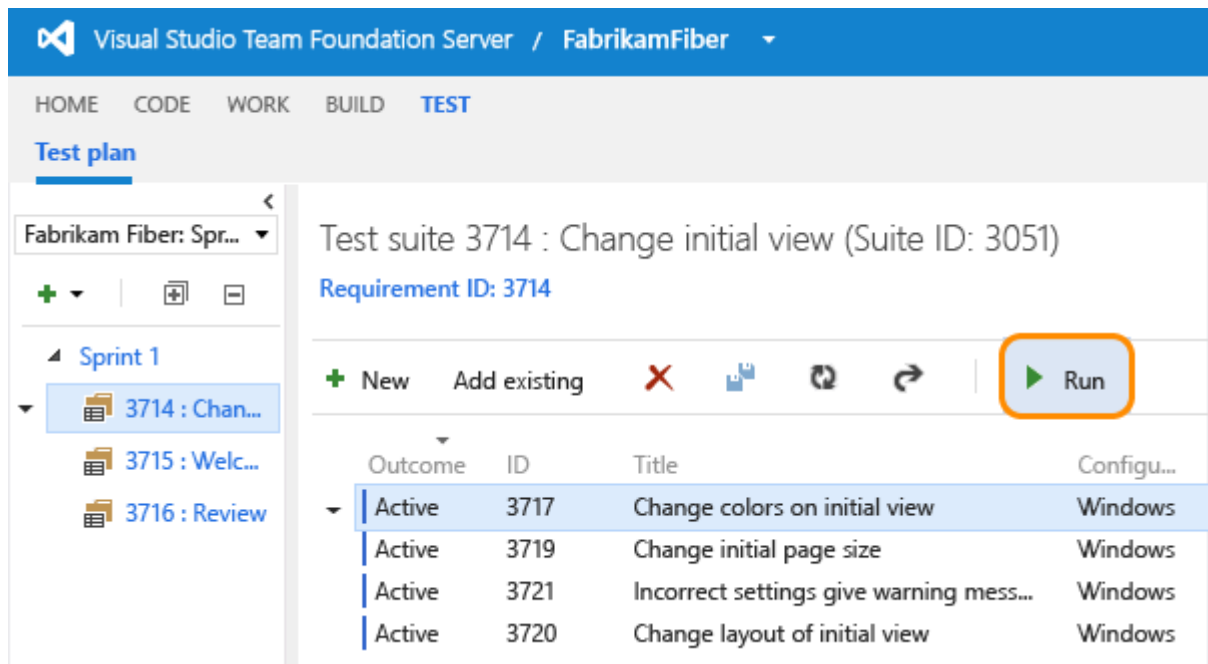
- Query Based: Use to group together test cases that have a particular characteristic - for example, all the tests that have Priority=1. The suite will automatically include every test case that is returned by the query that you define.
- Static: Use to group together test cases designed to track the test status of backlog items. Each test case that you add to a requirement-based test suite is automatically linked to the backlog item.
- Requirement Based: Use to group together test cases with any characteristics or test suites.

## 4.2 Running manual tests

Run your manual tests and record the test results for each test step using Microsoft Test Runner. If you find an issue when testing, use Test Runner to create a bug. Test steps, screenshots and comments are automatically included in the bug.

### 4.2.1 Run your manual tests

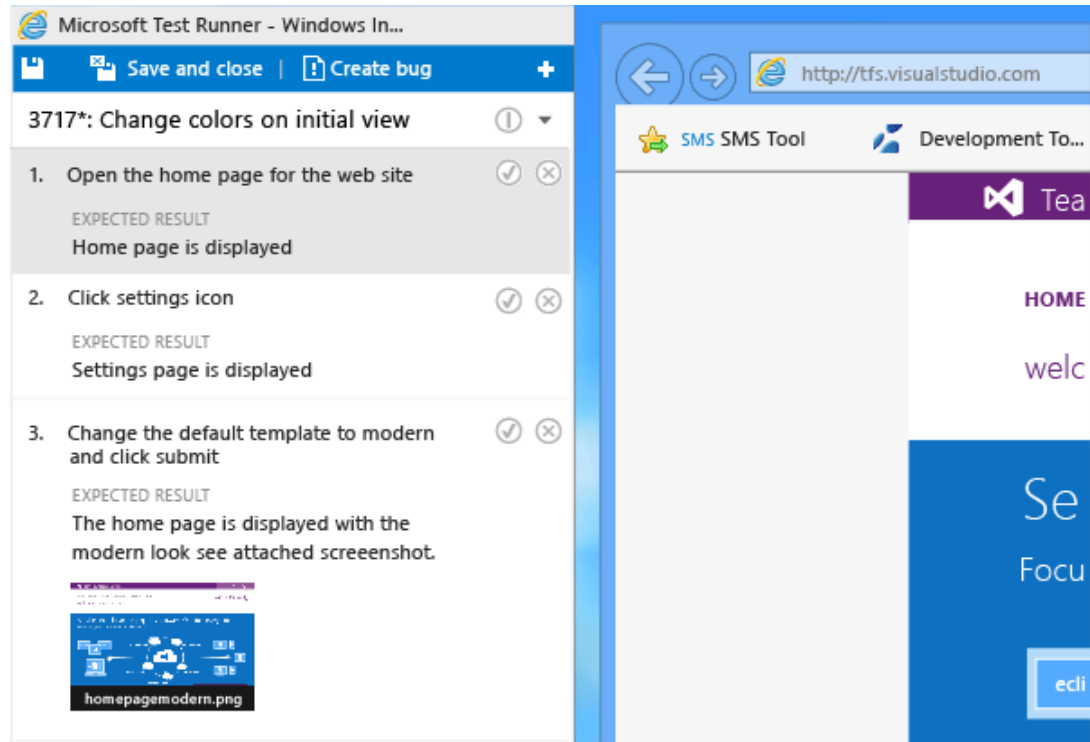
1. Select a test from a test suite and run it.



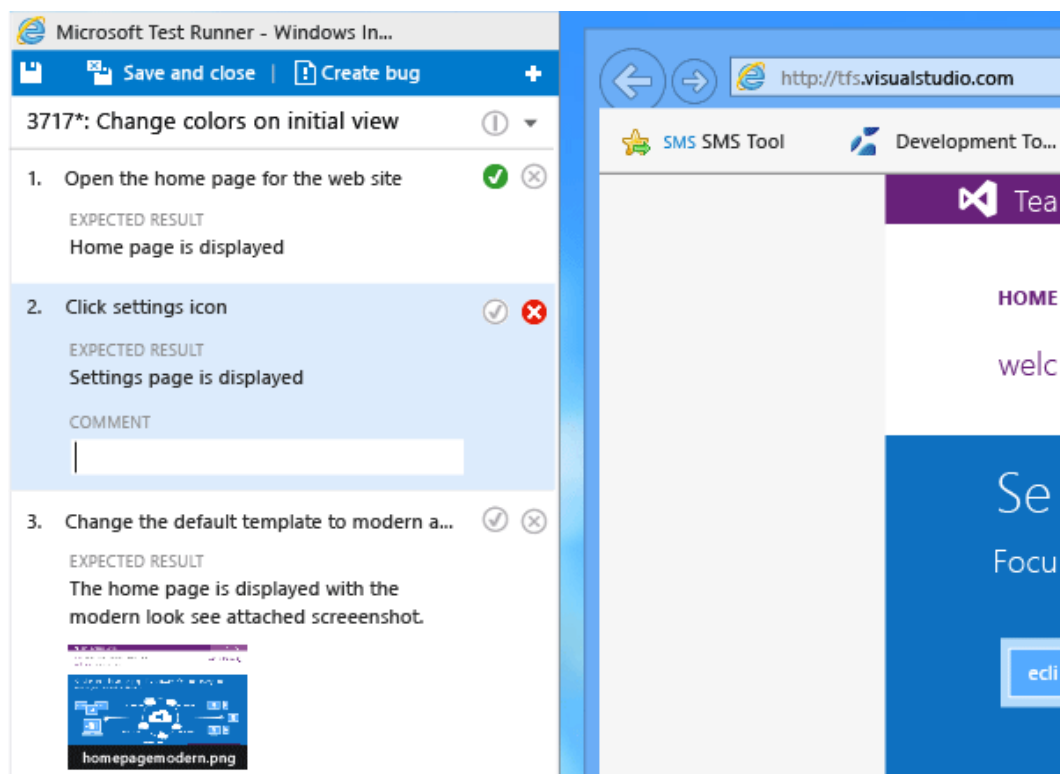
The screenshot shows the Visual Studio Team Foundation Server interface for a project named 'FabrikamFiber'. The 'TEST' tab is selected, and the 'Test plan' section is active. On the left, a tree view shows the test plan structure, including 'Sprint 1' and a test suite '3714 : Chan...'. The main area displays the details for 'Test suite 3714 : Change initial view (Suite ID: 3051)' with 'Requirement ID: 3714'. A toolbar at the top of the test suite area includes buttons for 'New', 'Add existing', and a 'Run' button (highlighted with an orange box). Below the toolbar is a table of test cases:

Outcome	ID	Title	Configu...
Active	3717	Change colors on initial view	Windows
Active	3719	Change initial page size	Windows
Active	3721	Incorrect settings give warning mess...	Windows
Active	3720	Change layout of initial view	Windows

2. Start the app that you want to test. Your app does not need to run on the same machine as Test Runner. You simply use Test Runner to record whether test steps pass or fail as you run a test manually. For example, you might run Test Runner on a desktop machine and run your Windows 10 store app that you are testing on a Windows 10 tablet.



- Mark each test step as either passed or failed based on the expected results. If a test step fails, you can enter a comment on why it failed.



- Create a bug to describe what failed.

Save and close

Create bug

+

3717\*: Change colors on initial view

1. Open the home page for the web site

EXPECTED RESULT

Home page is displayed

2. Click settings icon

EXPECTED RESULT

Settings page is displayed

COMMENT


Settings page is not dis...

3. Change the default tem...

EXPECTED RESULT

The home page is disp...

modern look see attac...



homepagemodern.png

New Bug 1\*: Field 'Title' cannot be empty.

Copy template URL

Tags

Add...

STATUS

Assigned To

<No one>

State

Active

Reason

New

CLASS

Area

Iterat...

REPRO STEPS

SYSTEM INFO

TEST CASES (1)

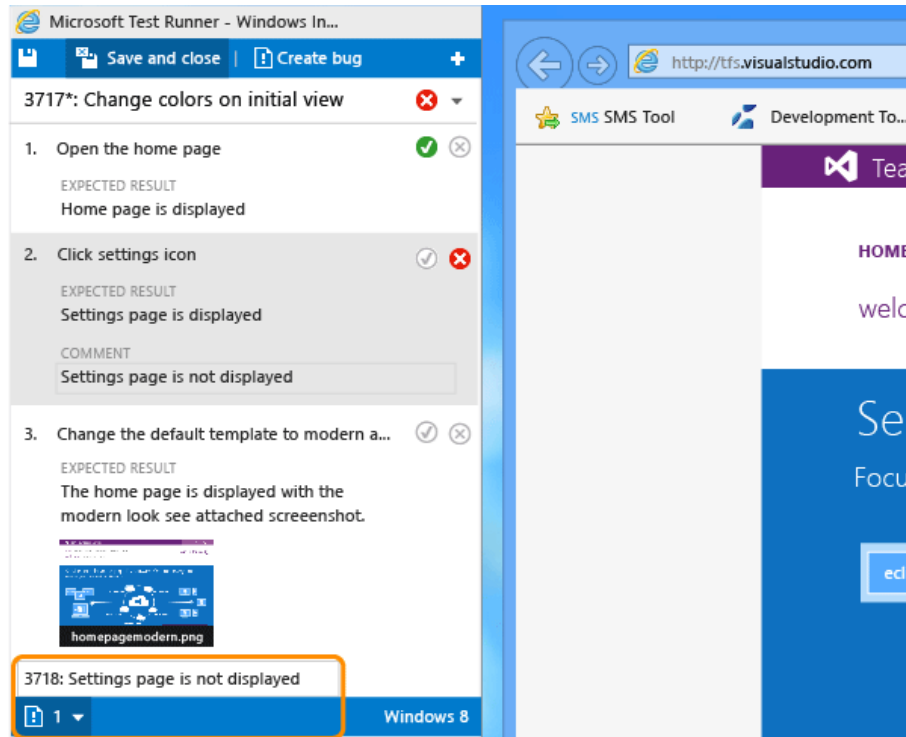
5/21/2013 11:31 AM

Bug filed on "Change colors on initial view"

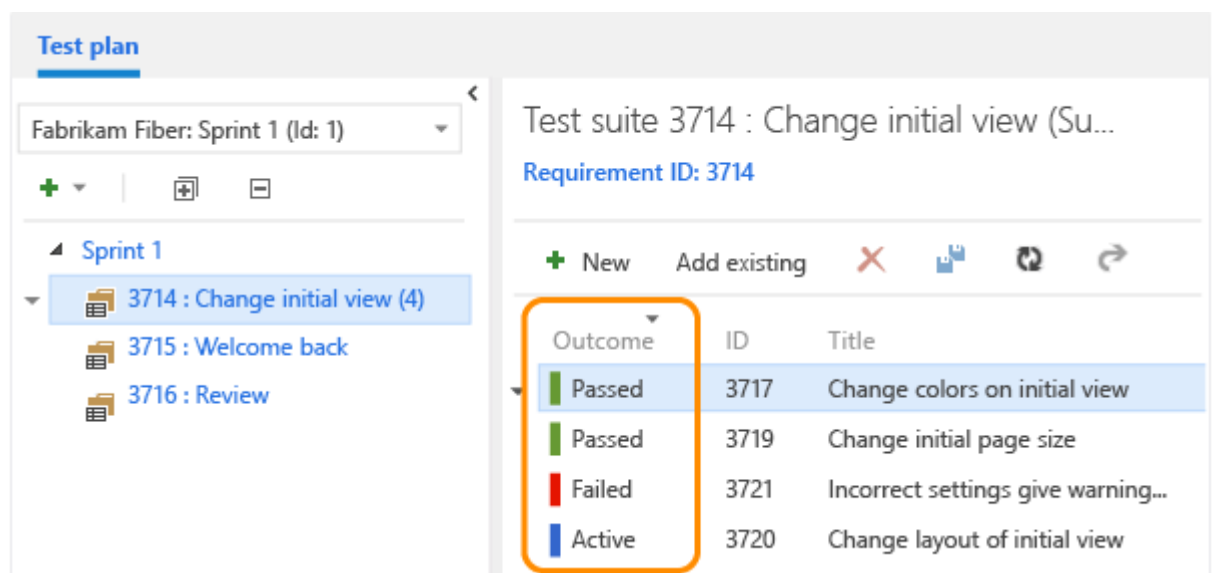
Step no.	Results	Title
1.	Passed	Open the home page for the web site Expected Result Home page is displayed
2.	Failed	Click settings icon Expected Result Settings page is displayed Comments: Settings page is not displayed

The steps and your comments are automatically added to the bug. Also, the test case is linked to the bug.

5. You can see any bugs that you have reported during your test session.



6. When you've run all your tests, save the results and close Test Runner. Now, all the test results are stored in TFS.
7. View the testing status for your test suite.



The most recent results for each test are displayed.

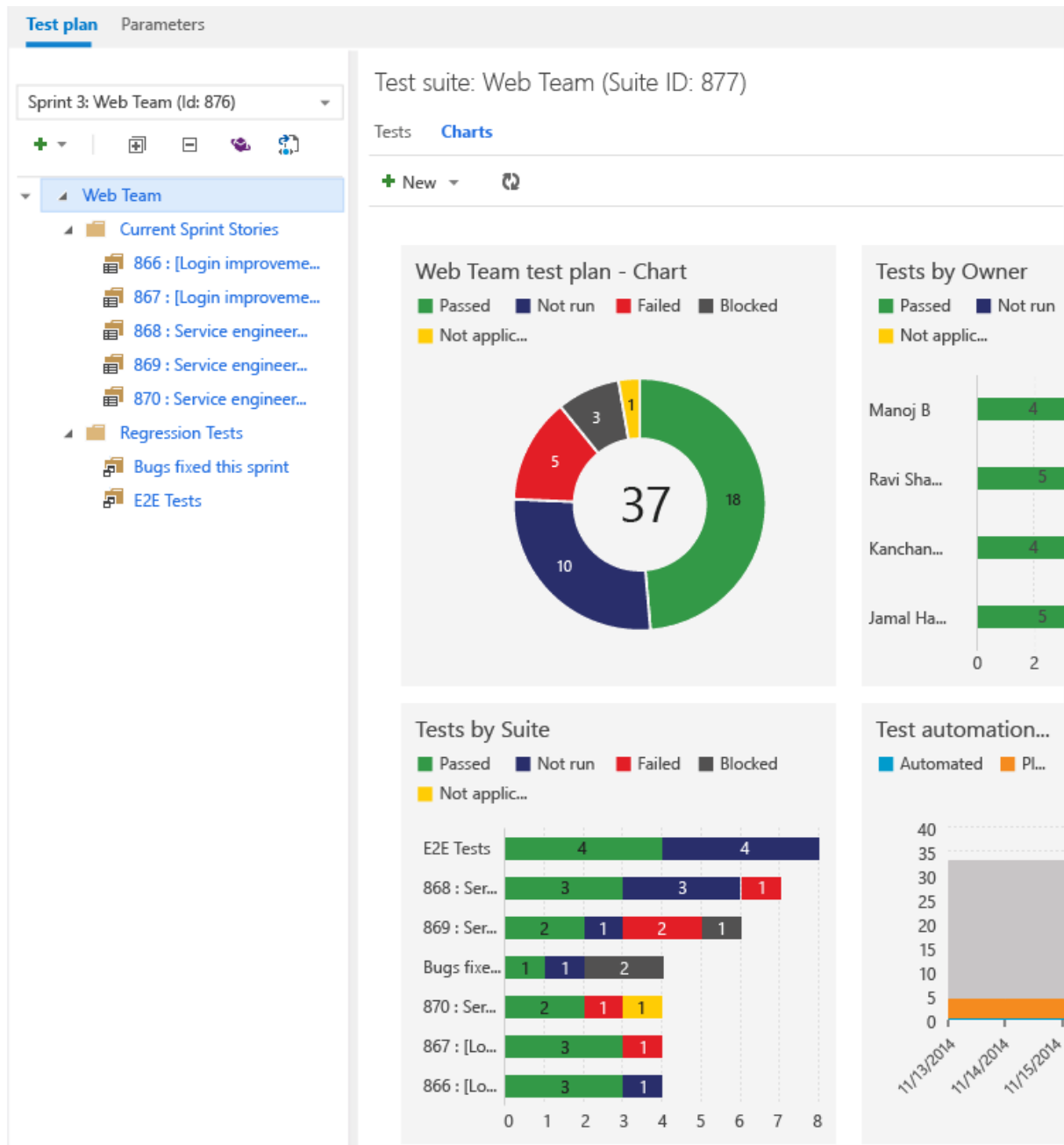
## 4.3 Track your test results

As you create and run tests, you want to be able to track your status. For example, how many of your tests are ready to run, or the pass/fail rate for tests, or how many tests have been run so far.

You can track your test status through the following reporting options:

- Lightweight charts (Team Web Access only)
- Work item queries (Team Web Access only)
- Pre-existing reports on warehouse data.
- Create your own reports from warehouse data.

Use lightweight charts to easily view the answers to many of your test status questions. You can pin these charts to your home page, then all the team can see the progress at a glance.



Create and run work item queries to generate reports from the work hub. From a flat-list work item query, you can also generate a chart for that query too. For example, create a query to show the active bugs and their priority and then generate a chart to quickly group these active bugs by priority.

But if lightweight charts don't give you the reports that you need, you can drill deeper into the data if you are using on-premises TFS. You can get access to existing reports on the warehouse data, or you can create your own with Excel.

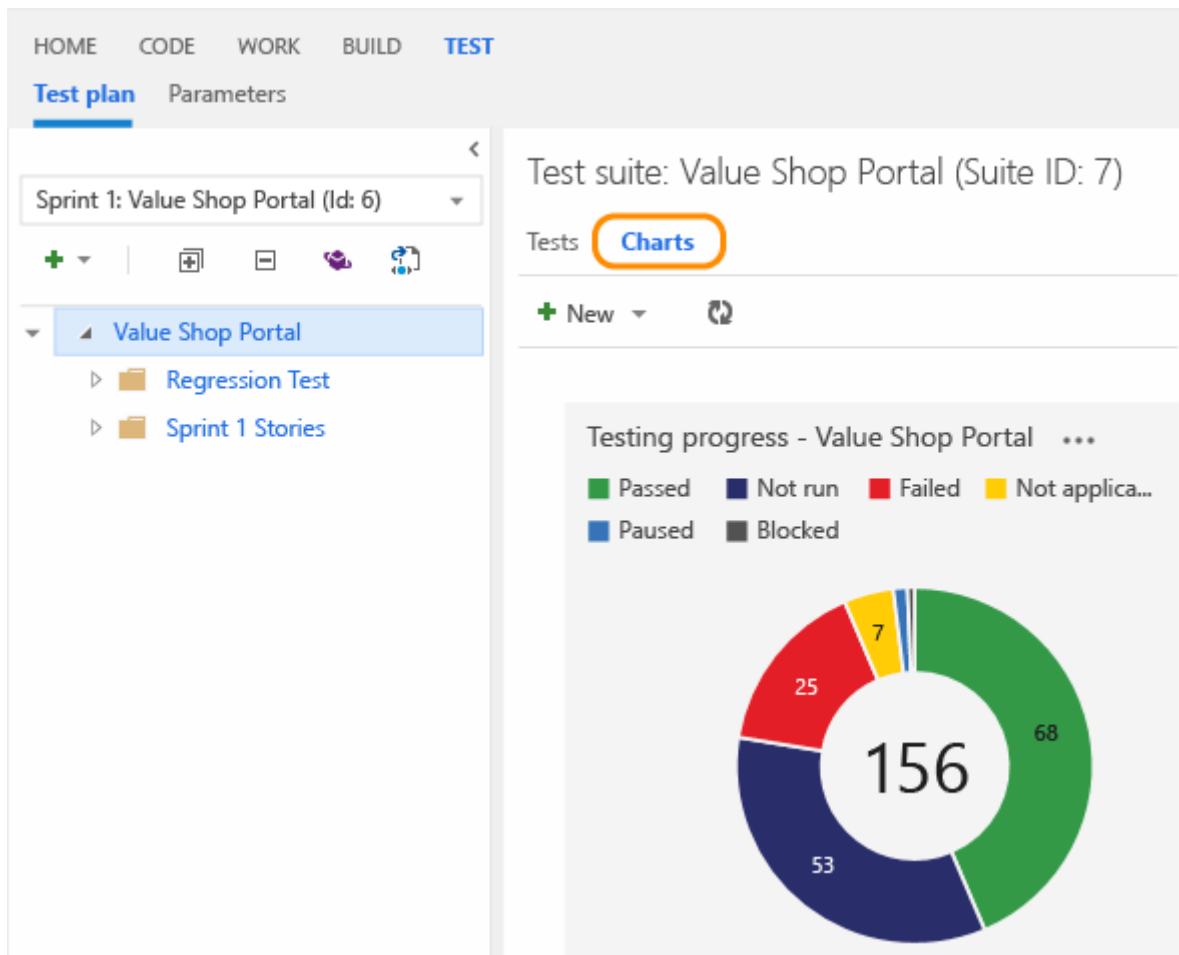
### 4.3.1 Test status with lightweight charts

You can create these charts from the test hub for Team Web Access only.

#### 4.3.1.1 Track testing progress

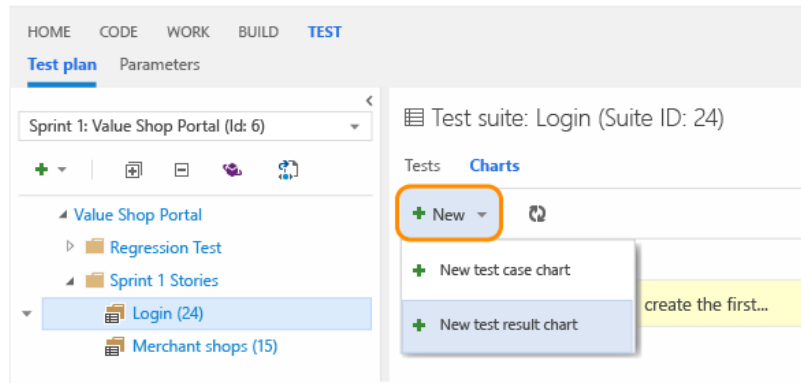
Use test results charts to track how your testing is going. Choose from a fixed set of pre-populated fields related to results. By default, a pie chart is created for each test plan. This chart is grouped by the outcome field to show the latest results for all the tests in the test plan.

View this default chart from the Charts tab.

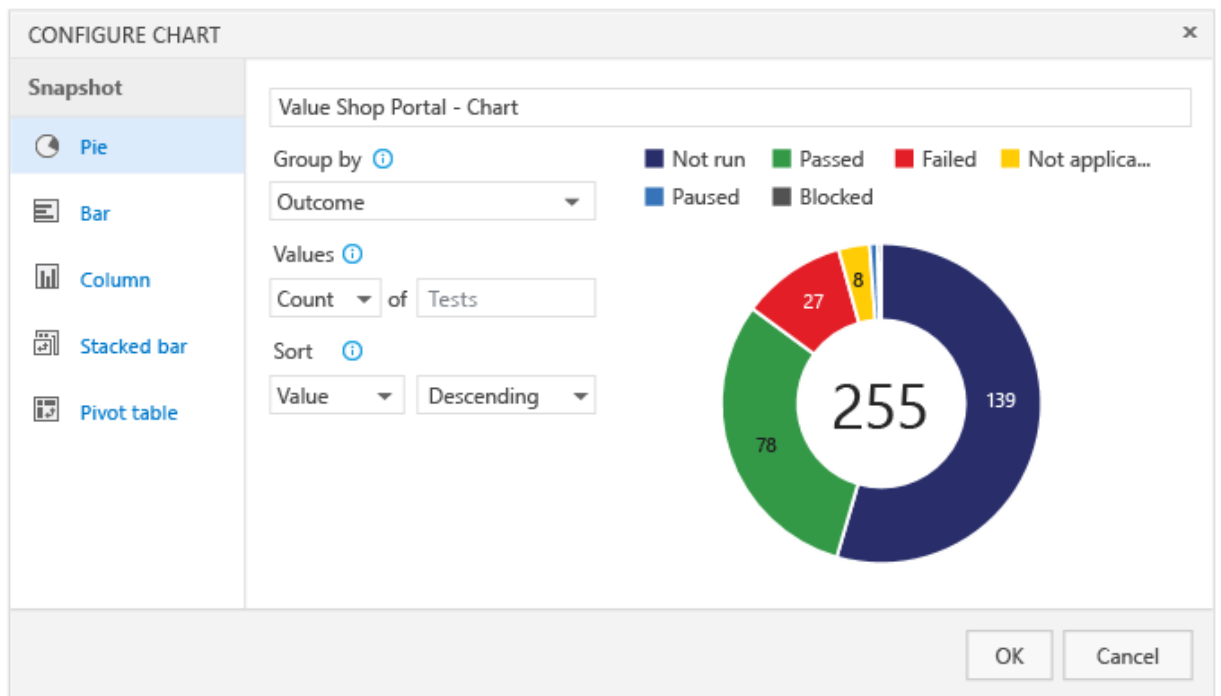


Add your own charts for test results to visualize what's important for your team as the following:

1. Select the test plan or test suite for your chart in the Test plan tab. Then create a new chart.



2. Select the chart type. Based on the chart, configure the fields that you want to use to group by, or for rows and columns.



All charts roll up the information for any child test suites of the test plan or test suite that you selected.

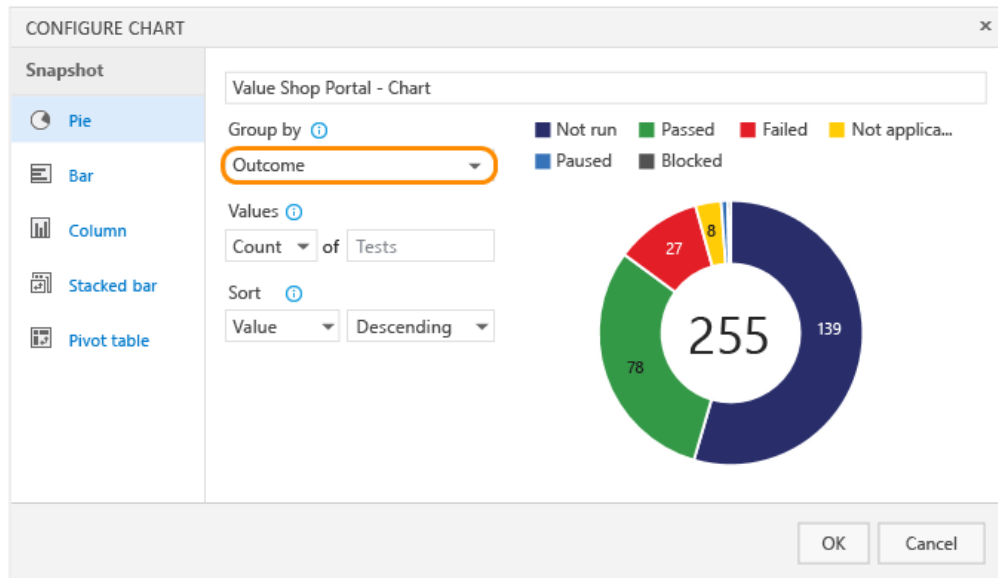
3. Save the chart. Now it will be displayed in the charts tab for the test plan or test suite that you selected.

#### 4.3.1.2 Test results examples

##### What's the test status for a specific test suite?

Select the test suite from the Test plan tab and add a test results pie chart. Group by outcome.

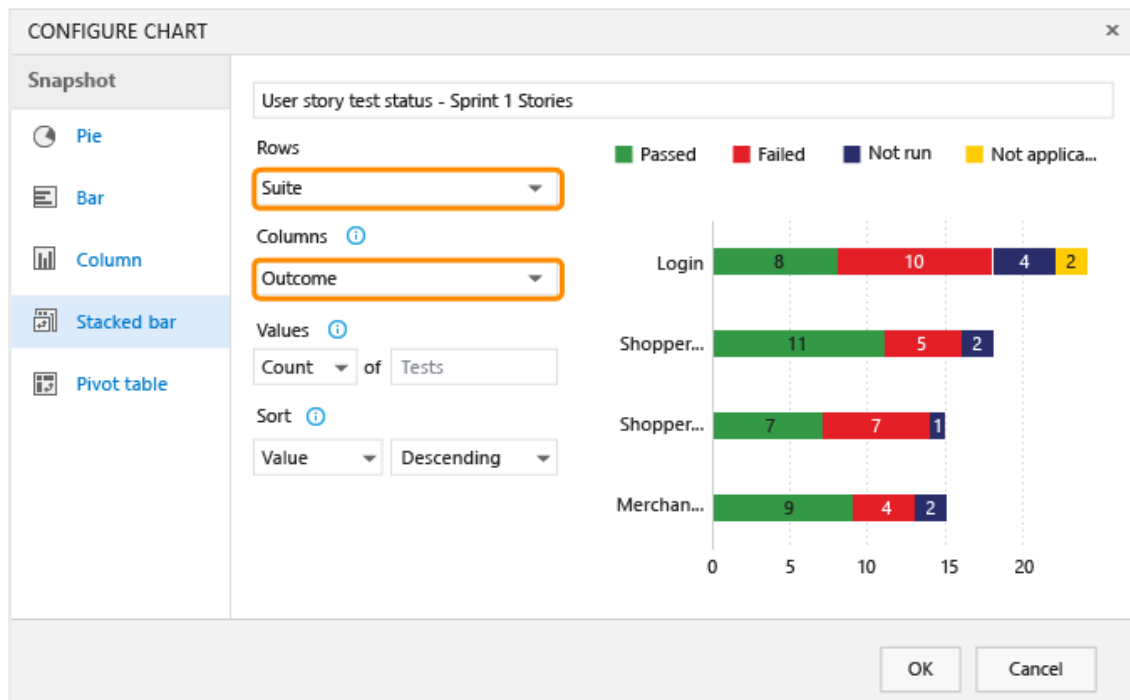




### What's the test status for user stories that my team's testing this sprint?

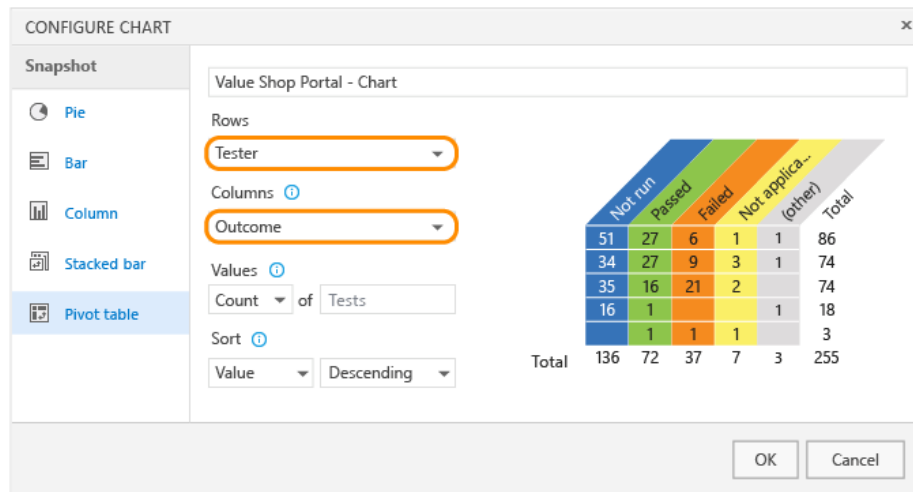
If you have created requirement-based test suites in your test plan for your user stories, you can create a chart for this.

1. Group these requirement-based test suites together in a static test suite.
2. Select this static test suite in the Test plan tab.
3. Add a test results stacked bar chart. Choose Suite as the rows pivot and Outcome as the columns pivot.



### How many tests has each tester left to run?

Select your test plan from the Test plan tab and add a test results pivot table chart. Choose Tester as the rows pivot and Outcome as the columns pivot.



How can I check quality based on the configuration?

Use either a stacked bar chart or a pivot table chart. Choose Configuration as the rows pivot and Outcome as the columns pivot.

How can I track why tests are failing for my team?

For failure analysis, use either a stacked bar chart or a pivot table chart. Choose Tester for the rows and Failure type for the columns. (Failure type for test results can only be set using Microsoft Test Manager.)

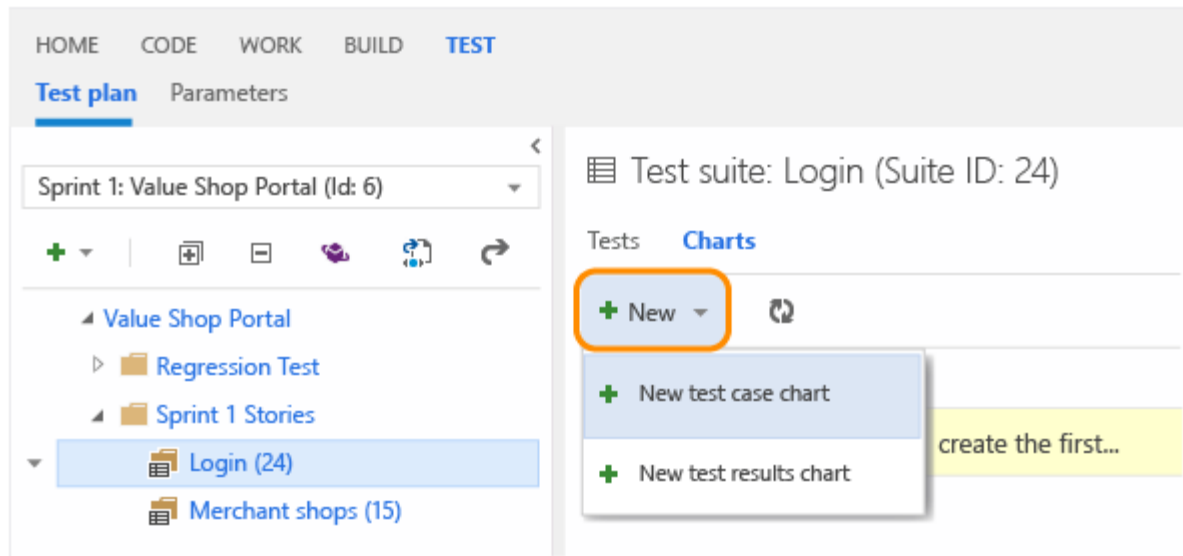
How can I track the resolution for failing tests for my team?

For resolution analysis, use either a stacked bar chart or a pivot table chart. Choose Tester for the rows and Resolution for the columns. (Resolution type for test results can only be set using Microsoft Test Manager.)

#### 4.3.1.3 Track test case status

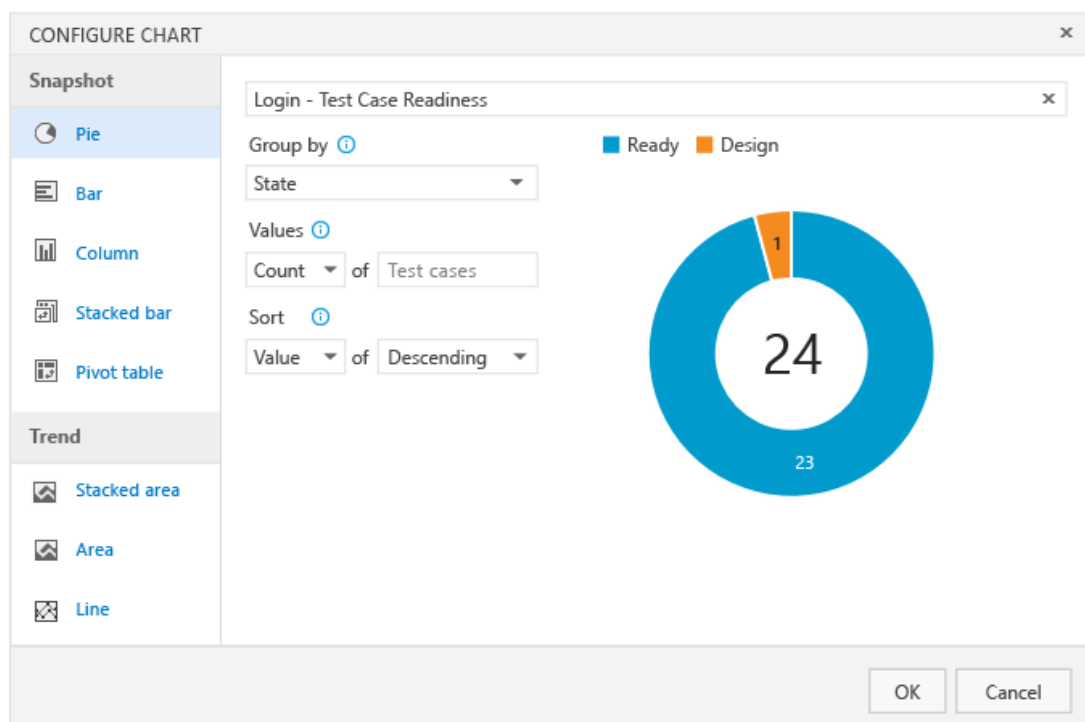
Use test case charts to find out the progress of your test case authoring. The charts for test cases give you the flexibility to report on columns that you add to the Tests tab. By default, test case fields are not added to the view in the Tests tab.

1. Add any fields you want to use for your test case chart from the Tests tab with Column options. Then the fields will appear as choices in the drop-down lists for grouping for your test case charts.
2. Select the test plan or test suite for your chart in the Test plan tab. Then add a test case chart.



All charts roll up the information for any child test suites of the test plan or test suite that you selected.

3. Select the chart type. Based on the chart, configure the fields that you want to use to group by, for rows and columns, or the range (trend charts only).



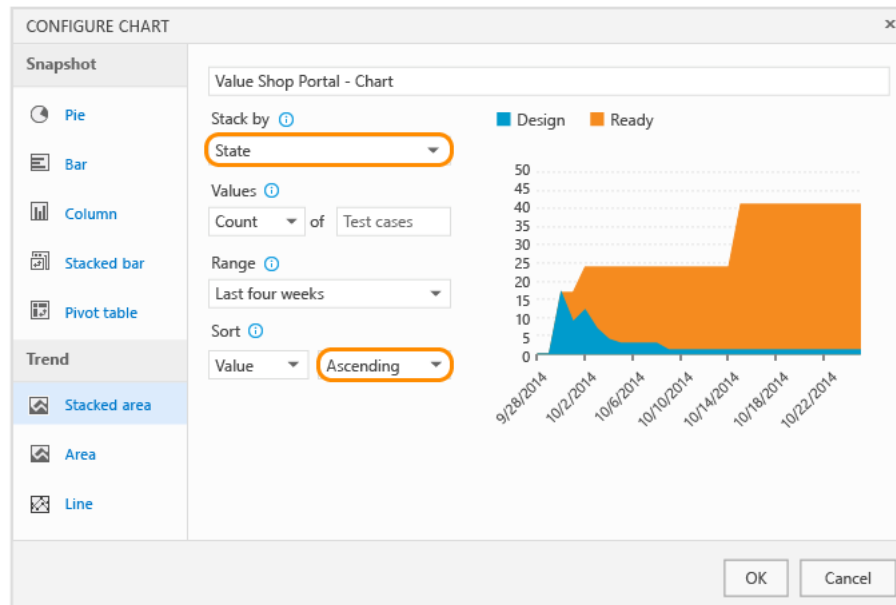
You cannot group by test suite for the test case charts.

4. Save the chart. Now it will be displayed in the charts tab for the test plan or test suite that you selected.

#### 4.3.1.4 Test cased examples

**How can I track burndown for test case creation?**

Use a stacked area trend chart to view the burndown for how many test cases are ready to be run. Choose State for the stack by field and Ascending for the sort field.



#### How can I track burndown for automation status?

Use a stacked area trend chart to view the burndown for how many test cases have been automated. Choose Automation status for the stack by field and Ascending for the sort field.

#### If multiple teams own test cases in my test plan, can I see how many each team owns and the priorities of the tests?

If your teams are organized by area path, then you can use a test case pie chart. Choose Area path for the group by field.

If you want to know the priorities of these tests, then create a stacked bar chart. Choose Area path for rows and priority for the columns.

#### How can I track test creation status by team members?

Test case owners are tracked by the Assigned to field. Use a stacked bar chart or a pivot table chart. Choose Assigned to for rows and status for the columns.

## 5 Bugs reporting and tracking

### 5.1 Capture bugs

How do you track and manage defects in your code? How do you make sure software problems and customer feedback get addressed in a timely manner to support high-quality software deployments? And, how do you do this while making good progress on new features?

At a minimum, you need a way to capture your software issues, prioritize them, assign them, and track progress. Moreover, you'll want to manage your bugs in ways that align with your practices.

You can track bugs in much the same way that you track the other work items. Using the bug work item form, you capture the code defect in the Title, Steps to reproduce, and other fields.

You can create bugs from the web portal, Visual Studio/Team Explorer, a work item template, or when testing with Test Manager.

The bug work item form tracks information bases on the methodology you follow as the following:

The screenshot shows a web portal interface for a bug report titled "Bug 364: Slow response on information form". The interface includes a title bar, a toolbar with icons for refresh, undo, redo, copy, and email, and a "Tags" section with an "Add..." button. The main form contains a title field with the text "Slow response on information form" and an ID field with the value "364". Below the title is an "Iteration" dropdown menu set to "Fabrikam Fiber\Release 1\Sprint 1". The form is divided into two main sections: "STATUS" and "DETAILS". The "STATUS" section includes fields for "Assigned To" (Raisa Pokrovskaya), "State" (Committed), "Area" (Fabrikam Fiber), and "Reason" (Commitment made by the team). The "DETAILS" section includes fields for "Priority" (2), "Severity" (3 - Medium), "Effort" (8), "Remaining Work", and "Activity". Below these sections are two tabs: "STEPS TO REPRODUCE" and "ACCEPTANCE CRITERIA". The "STEPS TO REPRODUCE" tab is active, showing a text area with a vertical cursor. The "ACCEPTANCE CRITERIA" tab is also visible, showing a text area. At the bottom of the form are three buttons: "Save", "Save and close", and "Close".

When defining a bug, use these fields described below to capture both the initial issue and ongoing discoveries made when triaging, investigating, fixing, and closing the bug.

Field/tab	Usage
Symptom	The unexpected behavior.
Proposed Fix	The proposed change to fix the reported
Steps to Reproduce (friendly name=Repro Steps)	<p>Capture enough information so that other team members can understand the full impact of the problem as well as whether they have fixed the bug. This includes actions taken to find or reproduce the bug and expected behavior.</p> <p>Describe the criteria that the team should use to verify whether the code defect is fixed.</p>
System Info	Information about the software and system configuration that is relevant to the test.
Acceptance Criteria	Provide the criteria to be met before the bug or user story can be closed. Before work begins, describe the customer acceptance criteria as clearly as possible. The acceptance criteria can be used as the basis for acceptance tests so that you can more effectively evaluate whether an item has been satisfactorily completed.
Found In Build	When Test Manager creates bugs, it automatically populates <b>System Info</b> and <b>Found in Build</b> with information about the software environment and build where the bug occurred. When you resolve the bug, use <b>Integrated in Build</b> to indicate the name of the build that incorporates the code that fixes the bug.
Integrated in Build	To access a drop-down menu of all builds that have been run, you can update the <b>FIELD</b> definitions for Found in Build and Integrated in Build to reference a global list. The global list is automatically updated with each build that is run.
Found in Environment	The software setup and configuration where the bug was found.
Root Cause	<p>The cause of the error. You can specify one of the following values:</p> <ul style="list-style-type: none"> <li>• Coding Error</li> <li>• Design Error</li> <li>• Specification Error</li> <li>• Communication Error</li> <li>• Unknown</li> </ul>
How Found	How the bug was found. For example, a bug might have been found during a customer review or through ad hoc testing.
Priority	<p>A subjective rating of the bug as it relates to the business or customer requirements. Priority indicates the order in which code defects should be fixed. You can specify the following values:</p> <ul style="list-style-type: none"> <li>• <b>1:</b> Product cannot ship without the successful resolution of the work item, and it should be addressed as soon as possible.</li> <li>• <b>2:</b> Product cannot ship without the successful resolution of the work item, but it does not need to be addressed immediately.</li> <li>• <b>3:</b> Resolution of the work item is optional based on resources, time, and risk.</li> </ul>
Severity	<p>A subjective rating of the impact of a bug on the project or software system. For example: If clicking a remote link (a rare event) causes an application or web page to crash (a severe customer experience), you might specify Severity = 2 - High and Priority = 3. Allowed values and suggested guidelines are:</p> <ul style="list-style-type: none"> <li>• <b>1 - Critical:</b> Must fix. A defect that causes termination of one or more system components or the complete system, or causes extensive data corruption. And, there are no acceptable alternative methods to achieve required results.</li> </ul>

	<ul style="list-style-type: none"> <li>• <b>2 - High:</b> Consider fix. A defect that causes termination of one or more system components or the complete system, or causes extensive data corruption. However, an acceptable alternative method exists to achieve required results.</li> <li>• <b>3 - Medium:</b> (Default) A defect that causes the system to produce incorrect, incomplete or inconsistent results.</li> <li>• <b>4 - Low:</b> A minor or cosmetic defect that has acceptable workarounds to achieve required results.</li> </ul>
--	--



Use the **History** section to add and review comments made about the work being performed to resolve the bug.

## 5.2 Track bugs as requirements or tasks

Many Development teams treat bugs the same as any requirement. Others see bugs as work that belongs to implementing a story, and therefore treat them as a task.

Bugs, like requirements, represent work that needs doing. So, should you track your bugs along with other items in the backlog items or as tasks linked to those backlog items? How does your team estimate work?

Based on how your team answers these questions, they can choose how they want to track bugs from one of these three choices:

Bug tracking options	Choose this option
<b>Bugs appear as part of the product backlog</b>  Bugs appear on backlogs and boards with requirements	When your team or product owner wants to manage bugs similar to requirements. Bugs can be added and prioritize along with PBIs or user stories on the product backlog.  With this option, the team can estimate the effort or story points for bugs which are then included against team velocity and cumulative flow.
<b>Bug backlog is separate from the product backlog</b>  Bugs appear on backlogs and boards with tasks	When your team links bugs to requirements, and manages them similar to tasks.  With this option, the team can estimate remaining work for bugs and track progress against the sprint capacity and sprint burndown.
<b>Bugs don't appear on backlogs and boards</b>	When your team manages bugs separate from requirements or tasks, or a different team is tasked with addressing bugs.

### 5.2.1 Triage bugs

Once you've started coding and testing, you'll want to hold periodic triage meetings to review and prioritize your bugs. How frequently you meet and for how long depends on your situation. Typically, the project owner runs the bug triage meetings, and team leads, business analysts and other stakeholders who can speak about specific project risks attend them.

The project owner can create or open a shared query for new and reopened bugs to generate a list of bugs to be triaged.

### 5.2.2 Bug queries

Open a shared query or use the query editor to create useful bug queries, such as the following:

- Active bugs by priority (**State <> Done** or **State <> Closed**)

- In Progress bugs (State = Committed or State = Active)
- Bugs to fix for a target release (Tags Contains RTM)
- Recent bugs - bugs opened within the last 3 weeks (Created Date > @Today-21)

Once you have the queries of interest to your team, you can create status or trend charts that you can also pin to a team dashboard.

### 5.2.3 Triage mode in query results

From the query results page, you can quickly move up and down within the list of bug work items using the up and down arrows. As you review each bug, you can assign it, add details, or set priority.

Active bugs

3 work items (1 selected)

Results

Editor

Charts

Work item pane Bottom

Save query

Column options

Copy query URL

ID	State	Assigned To	Title	Tags
364	Committed	Raisa Pokrovskaya	Slow response on information form	
384	Committed	Johnnie McLeod	Secure sign-in	Mobile Web
377	New		Switch context issues	

BUG 100

1 of 3

100

Slow response on form

New

Jamal Hartnett

2

Save

Area

Iteration

Updated by Jamal Hartnett

less than a minute ago

Fabrikam Fiber

Fabrikam Fiber\Iteration 1

Add Tag

Details

(3)

Repro Steps

Status

Reason

New defect reported

Build

Found in Build

Details

Priority

2

Severity

3 - Medium

Effort

### Tips for successful triage meetings:

Fixing bugs represents a trade-off with regards to other work. Use your triage meeting to determine how important fixing each bug is against other priorities related to meeting the project scope, budget, and schedule.

- Establish the team's criteria for evaluating which bugs to fix and how to assign priority and severity. Bugs associated with features of significant value (or significant opportunity cost of delay), or other project risks, should be assigned higher priority and severity. Store your triage criteria with other team documents and update as needed.
- Use your triage criteria to determine which bugs to fix and how to set their State, Priority, Severity, and other fields.



- Adjust your triage criteria based on where you are in your development cycle. Early on, you may decide to fix most of the bugs that you triage. However, later in the cycle, you may raise the triage criteria (or bug bar) to reduce the number of bugs that you need to fix.
- Once you've triaged and prioritized a bug, assign it to a developer for further investigation and to determine how to implement a fix.

#### **5.2.4 Pay down your bug debt**

Once bugs have been triaged, it's time to assign them to an iteration to get fixed. By addressing a set of bugs to get fixed every sprint, your team can keep the total number of bugs to a reasonable size.

When bugs appear on the backlog, you can assign bugs to iterations in the same way you do requirements during your iteration planning sessions.

When bugs are treated as tasks, they're often automatically linked to a requirements. So, assigning their parent requirement to an iteration will assign the linked bugs to the same iteration as the parent requirement during your iteration planning sessions.

Your team should consider fixing all bugs found during an iteration when testing a feature in development.

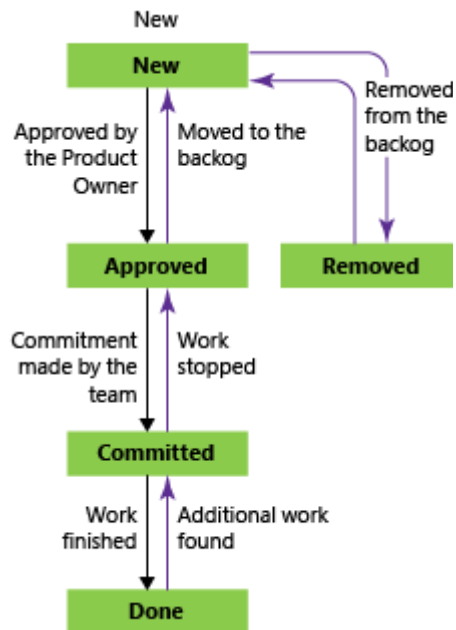
#### **5.2.5 Fix, resolve and close bugs**

Bug fixes that involve more than a single section of code may require significant regression testing and may involve other team members. Record any conversations that relate to assessing the risk of bug fixes in the bug work item history.

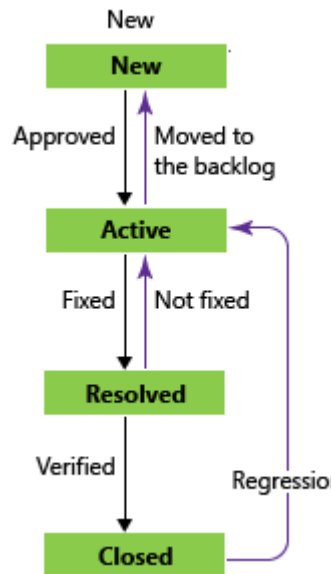
Once you fix a bug, you should update its State. State choices vary depending on the process you use--Scrum, Agile, or CMMI.

For Scrum bugs, you simply change the State to Done. For Agile and CMMI, you first resolve the bug. Typically, the person who created the bug then verifies the fix and sets the State to Closed.

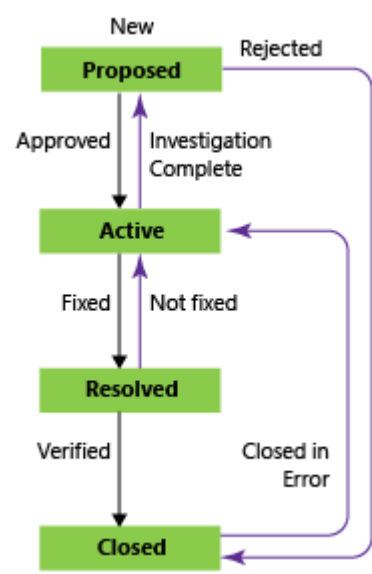
## Scrum



## Agile



## CMMI



If more work has been found after a bug has been resolved or closed, it can be reactivated by setting the State to Committed or Active.

### 5.2.6 Verify a fix

To verify a fix, a developer or tester should attempt to reproduce the bug and look for additional unexpected behavior. If necessary, they should reactivate the bug.

When verifying a bug resolution, you may find that the bug was not completely fixed or you may disagree with the resolution. In this case, discuss the bug with the person who resolved it, come to an agreement, and possibly reactivate the bug. If you reactivate a bug, include the reasons for reactivating the bug in the bug description.

### 5.2.7 Close a bug

You close a bug once it's verified as fixed. However, you may also close a bug for one of these reasons:

- Deferred - deferring a fix until the next product release
- Duplicate - bug has already been reported, provide a link to the bug which remains open
- As Designed - feature works as designed
- Cannot Reproduce - tests prove that the bug can't be reproduced
- Obsolete - the bug's feature is no longer in the product
- Copied to Backlog - a PBI or user story has been opened to track the bug

It's always a good idea to describe any additional details for closing a bug in the History field to avoid future confusion as to why the bug was closed.

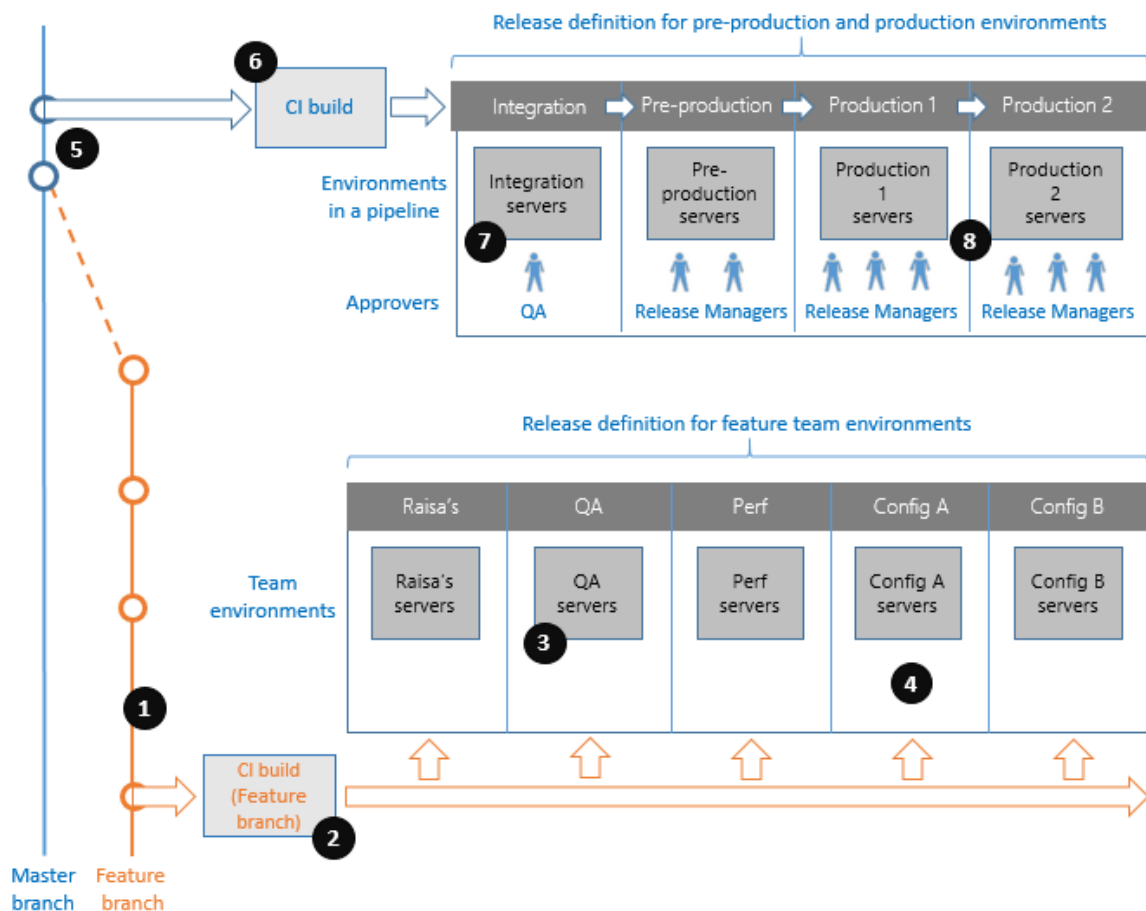
## 6 Build and release management

### 6.1 Understanding Microsoft release management

Release Management is a service in Visual Studio Team Services and Team Foundation Server (update 2 and later) that helps you automate the deployment and testing of your software in multiple environments. Using Release Management, you can either fully automate the delivery of your software all the way to production, or set up semi-automated processes with approvals and on-demand deployments. It is an essential element of DevOps that helps your team continuously deliver software to your customers at a faster pace and with lower risk.

### 6.2 Release management workflow

During an iteration, members of the development team pick up backlog items and commit their changes to a Feature branch (1). A continuous integration build continuously integrates the changes in your Feature branch (2) as the following:



Developers want new releases to be regularly deployed to a **Team QA environment** (3), where automated functional tests are run. In addition, they want to have a number of environments set up regularly with the latest code for **manual, performance, or multi-configuration testing** (4).

At periodic intervals throughout the Sprint, your team integrates the **Feature** branch with the **Master** branch (5). Related applications developed by other teams are constantly integrated into the master branch as well.

To ensure that the commits into that branch are of high quality, a CI build of the master branch runs for each check-in (6).

You want to deploy every new release in the master branch to an **Integration** environment and run functional tests (7). Your operations team manages a **Pre-production** environment and a series of **Production** environments. Early adopters of your applications are hosted in the first **Production** environment, while others are hosted in later **Production** environments (8).

At the end of every Sprint, the operations team wants to take a release that was successfully deployed to the **Integration** environment, and promote it through **Pre-production** and **Production** environments in turn. Upgrade tests are run in the **Pre-production** environment to minimize problems that might arise in the **Production** environment. You rely on feedback from early adopters to decide whether a release should be promoted from one **Production** environment to another.

Your release managers want to track the progress of all deployments. When a deployment fails, or when tests fail, developers need access to logs for immediate diagnosis. And your operations teams must be able to roll back a deployment if the fix for an issue will take too long.

When an older release is to be deployed, release managers need to understand all the features that will be rolled back. When the fix is finally ready, you want to track it all the way to **Production**, and you need to confirm that the issue your customers encountered is, indeed, fixed.

The application development and release process in your own organization is likely to have some variations from this example, though it is also likely to have quite a lot in common. In fact you may have many applications flowing through similar release pipelines in parallel, with the typical requirements demonstrated in this use case. Release Management helps your team and organization fulfill such requirements with ease, as described in the typical workflow shown in the next section of this topic.

When using Release Management, here are the steps that you typically follow:

1. **Create release definitions:** You start using Release Management by creating a release definition in the RELEASE hub of your team project. A release definition specifies (a) What to deploy - the set of artifacts that constitutes a new release, and (b) How to deploy - the series of automation tasks that should be run in each environment.
2. **Add environments:** You add one or more environments to a release definition. Each environment is simply a named logical entity that represents a deployment target for your release. For example, you create environments for test, quality assurance, staging, and production. You then edit the environments to specify the lists of users that must approve the deployment where this is appropriate.
3. **Add tasks:** You add automation tasks to each environment. These tasks describe the deployment and testing process. There is a wide range of pre-defined tasks you can use. These tasks can take advantage of shared custom variables and built-in properties in their configuration. Your tasks may need to connect to other services, cloud platforms, or third party deployment and testing services. For this, you define your global service endpoints.
4. **Create and deploy releases:** Once you have created a release definition, you can manually create a new release based on this definition and deploy it to various environments, or you can let a release be created automatically and deployed upon completion of a build. There is a wide range of options for creating a release and deploying it, including release date and time, pipeline or parallel deployment, and more. You can also monitor the new changes that went into each release.
5. **Track deployments:** As a release is deployed to various environments, you track its progress. You can approve new deployments to an environment, and view the logs of deployments as they happen.

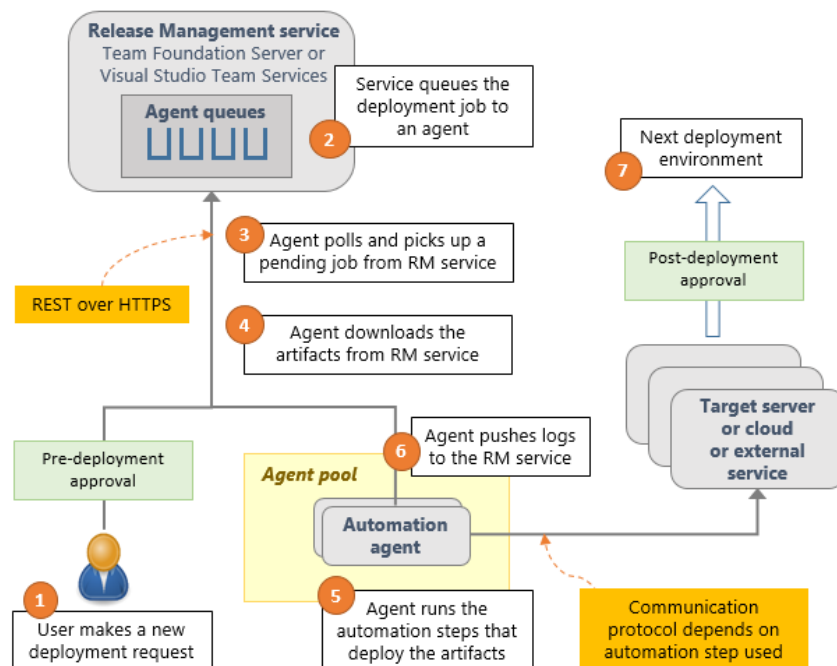


It's important to recognize the difference between a **release** and a **deployment**. A **release** is simply a package or container that defines the set of versioned artifacts and other information for a specific set of deployments. It's basically a snapshot of the release definition at the point the release is created, allowing the release definition to be modified independently from the release. A release definition can be used to create multiple releases. Each release generates a **deployment** to one or more environments by executing tasks. Deployments can be initiated manually or orchestrated automatically, as part of a pipeline of parallel or sequential deployments.



### How does Release Management work?

The Release Management service stores the data about your release definitions, environments, tasks, releases, and deployments in Team Services or Team Foundation Server.



Release Management runs the following steps as part of every deployment:

- 1. Pre-deployment approval:** When a new deployment request is triggered, Release Management checks whether a pre-deployment approval is required before deploying a release to an environment. If it is required, it sends out email notifications to the appropriate approvers.
- 2. Queue deployment job:** Release Management schedules the deployment job on an available automation agent. An agent is a piece of software that is capable of running tasks in the deployment.

3. **Agent selection:** An automation agent picks up the job. The agents for Release Management are exactly the same as those that run your Builds in Team Services and Team Foundation Server. A release definition can contain settings to select an appropriate agent at runtime.
4. **Download artifacts:** The agent downloads all the artifacts specified in that release (provided you have not opted to skip the download). The agent currently understands two types of artifacts: Team Build artifacts and Jenkins artifacts.
5. **Run the deployment tasks:** The agent then runs all the tasks in the deployment job to deploy the app to the target servers for an environment.
6. **Generate progress logs:** The agent creates detailed logs for each step while running the deployment, and pushes these logs back to Team Services or Team Foundation Server.
7. **Post-deployment approval:** When deployment to an environment is complete, Release Management checks if there is a post-deployment approval required for that environment. If no approval is required, or upon completion of a required approval, Release Management proceeds to trigger deployment to the next environment.

### 6.2.1 Create release definitions

A **release definition** is one of the fundamental concepts in Release Management. A release definition defines:

- The types and sources of **artifacts** that make up new releases.
- The collection of **environments** in which the artifacts can be deployed.
- The **automation tasks** that can be executed in each environment.

The following describes some of the general features and options for release definitions:

#### 6.2.1.1 Global configuration properties

You can define configuration properties as custom variables that are global to a release definition, and use these to specify values for any of the tasks in any of the environments of a release definition. You need to update the value in only a single location in order to update all the places where it is used.

- Use a **global configuration property** in a release definition when you need to use the same value across all the environments and you need to change the value in a single place.
- Use an **environment-level configuration property** for values that vary from environment to environment.

You can designate a configuration property to be a secure property by selecting the padlock icon next to the property. The values of such properties are stored securely on the server and cannot be viewed by users once they are saved. During a deployment, the Release Management service decrypts those values that are referenced by the tasks and passes them to the agent over secure HTTPS channel.

Definition\*: Purchasing | Releases

Environments Artifacts **Configuration** Triggers General History

Release variables

Environment variables

Release variables

Define custom variables to use in this release definition. View list of [pre-defined variables](#)

Name	Value
AzureContainerUri	.....
AzureStorageAccount	fabrikaminfra
AzureStorageContainer	purchasing

+ Add variable

You can also use the built-in properties of Release Management in the tasks of your release definition.

You can also view and edit the individual configuration properties for each environment in the release definition in this tab, as well as in the **Variables** dialog of an environment.

Definition: FabrikamDev | Releases

Environments Artifacts **Configuration** Triggers General

Release variables

Environment variables

Release variables

View and modify custom variables defined on the environments in this release definition.

	Dev	Test	Production
AdministratorLogin			
AdministratorLoginPassword	*****	*****	*****
ConnectionStringName			
DatabaseName			
ReleaseConfiguration	Release	Release	Release
ReleasePlatform	Any CPU	Any CPU	Any CPU

#### 6.2.1.2 Continuous deployment/integration and scheduled triggers

a release definition can be configured to automatically create a new release when it detects new artifacts are available - typically as a result of a new build of the application.

This automatic behavior is configured by the settings in the **Triggers** tab of a release definition.

Definition: FabrikamDev | Releases

EnvironmentsArtifactsConfigurationTriggersGeneralHistory

Refresh | Save | + Release

☒ **Manual**  
Create a new release manually using the "Release" action.

☐ **Continuous Deployment**  
Creates a new release every time a new artifact version is available.

☐ **Scheduled**  
Create a new release at a specified time.

**Environment triggers**

Environment triggers specify when and how a deployment will be triggered on an environment.

Environment	Trigger	
Dev	Manual	
QA	Manual	
Production	Automated: after release creation	

The upper section of the tab provides three options:

- **Manual:** No releases are initiated automatically when a new build of the source artifacts occurs. All releases for this release definition must be created manually by choosing the **Release** icon in a release definition or from a build summary..
- **Continuous Deployment:** A new release is created automatically when Release Management detects new built artifacts are available. When you select this option, a drop-down list enables you to select which of the **artifact sources** linked to this release definition will trigger a new release. The name of these new releases is configured in the **release name policy**.

☐ **Manual**  
Create a new release manually using the "Release" action.

☒ **Continuous Deployment**  
Creates a new release every time a new artifact version is available.  
Set trigger on artifact source  
FabrikamDev

☐ **Scheduled**  
Create a new release at a specified time.

**Scheduled:** A new release is created based on a schedule you specify. When you select this option, a set of controls enables you to select the days of the week and the time of day that Release Management will automatically create a new release. The name of these new releases is configured in the **release name policy**.



☐ **Manual**  
 Create a new release manually using the "Release" action.




☐ **Continuous Deployment**  
 Creates a new release every time a new artifact version is available.


☒ **Scheduled**  
 Create a new release at a specified time.

24h time:  :  (UTC) Coordinated Universal Time

Su ☐ M ☒ Tu ☒ W ☒ Th ☒ F ☒ Sa ☐

However, even though a release is automatically created, it **might not be deployed automatically** to an environment. To enable automatic deployment, you must also configure **environment deployment triggers** in each environment for which you want automated deployments to occur. The lower section of the **Triggers** tab lists the environments configured for this release definition.

Environment triggers		
Environment triggers specify when and how a deployment will be triggered on an environment.		
Environment	Trigger	
Dev	Manual	
QA	Manual	
Production	Automated: after release creation	




The  icon for each one opens the **Deployment conditions** dialog for that environment, which shows the currently configured environment deployment triggers and deployment queuing policies. Users with permission to edit release definitions can edit the deployment conditions here, including environment deployment triggers and deployment queuing policies. However, these settings can also be configured directly for each environment.

### 6.2.1.3 Release names


The names of releases for a release definition are, by default, sequentially numbered. The first release is named **Release-1**, the next release is **Release-2**, and so on. You can change this naming scheme by editing the release name format mask. In the **General** tab of a release definition, edit the **Release name format** property.

Definition\*: FabrikamDev | [Releases](#)

[Environments](#) | [Artifacts](#) | [Configuration](#) | [Triggers](#) | **[General](#)** | [History](#)

 |  Save |  Release ▼

Define the naming scheme for releases of this release definition. View the list of available [pre-defined variables](#).

Release name format  

When specifying the format mask, you can use the following pre-defined variables.

Variable	Description
Rev:rr	An auto-incremented number with at least the specified number of digits.
Date / Date:MMddyy	The current date, with the default format <b>MMddyy</b> . Any combinations of M/MM/MMM/MMMM, d/dd/ddd/dddd, y/yy/yyyy/yyyy, h/hh/H/HH, m/mm, s/ss are supported.
System.TeamProject	The name of the team project to which this build belongs.
Release.Releaseld	The ID of the release, which is unique across all releases in the project.
Release.DefinitionName	The name of the release definition to which the current release belongs.
Build.BuildNumber	The number of the build contained in the release. If a release has multiple builds, this is the number of the build that triggered the release in the case of continuous deployment, or the number of the first build in the case of a manual trigger.
Build.DefinitionName	The definition name of the build contained in the release. If a release has multiple builds, this is the definition name of the build that triggered the release in the case of continuous deployment, or the definition name of the first build in the case of manual trigger.
Artifact.ArtifactType	The type of the artifact source linked with the release. For example, this can be <b>Team Build</b> or <b>Jenkins</b> .
Build.SourceBranch	The branch for which the build in the release was queued. For Git, this is the name of the branch in the form <b>refs/heads/master</b> . For Team Foundation Version Control, this is the root server path for the workspace in the form <b>\$/teamproject/branch</b> . This variable is not set in the case of Jenkins artifact sources.
<i>Custom variable</i>	The value of a global configuration property defined in the release definition.

#### 6.2.1.4 Release names

The names of releases for a release definition are, by default, sequentially numbered. The first release is named **Release-1**, the next release is **Release-2**, and so on. You can change this naming scheme by editing the release name format mask. In the **General** tab of a release definition, edit the **Release name format** property.

Definition\*: FabrikamDev | Releases

[Environments](#)
[Artifacts](#)
[Configuration](#)
[Triggers](#)
[General](#)
[History](#)

| Save | Release ▼

Define the naming scheme for releases of this release definition. View the list of available [pre-defined variables](#).

Release name format

When specifying the format mask, you can use the following pre-defined variables.

Variable	Description
Rev:rr	An auto-incremented number with at least the specified number of digits.
Date / Date:MMddyy	The current date, with the default format <b>MMddyy</b> . Any combinations of M/MM/MMM/MMMM, d/dd/ddd/dddd, y/yy/yyyy/yyyy, h/hh/H/HH, m/mm, s/ss are supported.
System.TeamProject	The name of the team project to which this build belongs.
Release.ReleaseId	The ID of the release, which is unique across all releases in the project.
Release.DefinitionName	The name of the release definition to which the current release belongs.
Build.BuildNumber	The number of the build contained in the release. If a release has multiple builds, this is the number of the build that triggered the release in the case of continuous deployment, or the number of the first build in the case of a manual trigger.
Build.DefinitionName	The definition name of the build contained in the release. If a release has multiple builds, this is the definition name of the build that triggered the release in the case of continuous deployment, or the definition name of the first build in the case of manual trigger.
Artifact.ArtifactType	The type of the artifact source linked with the release. For example, this can be <b>Team Build</b> or <b>Jenkins</b> .
Build.SourceBranch	The branch for which the build in the release was queued. For Git, this is the name of the branch in the form <b>refs/heads/master</b> . For Team Foundation Version Control, this is the root server path for the workspace in the form <b>\$/teamproject/branch</b> . This variable is not set in the case of Jenkins artifact sources.
Custom variable	The value of a global configuration property defined in the release definition

#### 6.2.1.5 Release retention

The release retention setting controls how long a release is retained. By default, this is 60 days, and so releases that have not been deployed or modified during that time will automatically be deleted.

You can configure how long releases are retained by setting a retention value in days for each release definition. In the **General** tab of a release definition, edit the **Release retention policy** property. The maximum value is 365 days.

The screenshot shows the 'General' tab of a release definition in Release Management. The title bar indicates the definition is 'FabrikamDev'. Below the title bar are tabs for 'Environments', 'Artifacts', 'Configuration', 'Triggers', 'General' (which is selected and highlighted with an orange border), and 'History'. Below the tabs is a toolbar with a refresh icon, a 'Save' button, and a '+ Release' button. The main content area contains two input fields. The first is 'Release name format' with the value 'Release-\$(rev:r)'. The second is 'Release retention policy' with the value '60'. Both input fields have an information icon (i) to their right. The 'Release retention policy' field is highlighted with an orange border.

#### 6.2.1.6 Security

Security in Release Management follows the same model as Team Foundation security. *Permissions* define the authorizations that can be granted or denied to users and groups. These permissions can be granted or denied in a hierarchical model at the team project level, for a specific release definition, or for a specific environment in a release definition. Within this hierarchy, permissions can be inherited from the parent or overridden.

Release Management defines a Team Services group named **Release Administrators** at project level. You can add project members to this group, and set permissions for group members, in the main Control Panel page, on the **Security** tab.

Control panel > DefaultCollection > Fabrikam

Overview Iterations Areas **Security** Alerts Version Control Service Hook

Create VSTS group

Filter users and groups

- Teams
  - FabrikamDev Team
- VSTS Groups
  - Build Administrators
  - Contributors
  - Endpoint Administrators
  - Endpoint Creators
  - Project Administrators
  - Project Valid Users
  - Readers
  - Release Administrators**

FabrikamDev > Release Administrators | Edit...

**Permissions** Members Member of

Members of this group can perform all operations on Release Management

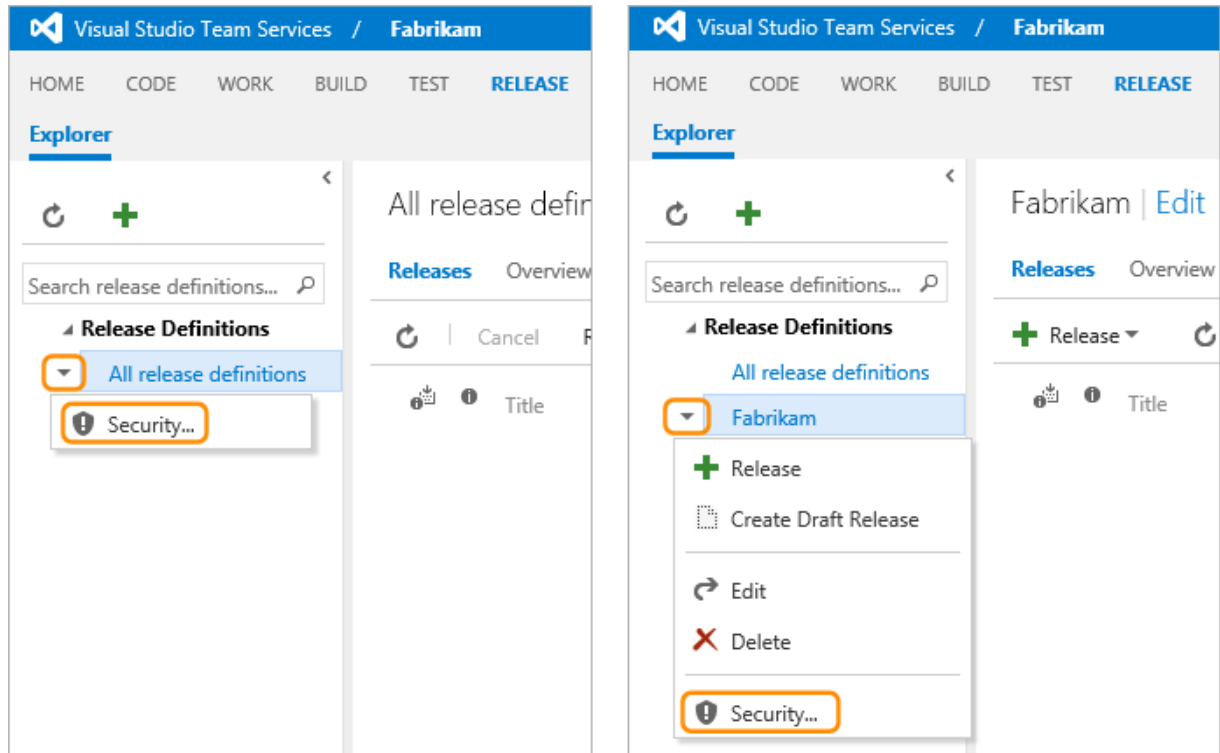
Create tag definition	Not set
Create test runs	Not set
Delete team project	Not set
Delete test runs	Not set
Edit project-level information	Not set
Manage test configurations	Not set
Manage test environments	Not set
View project-level information	Not set
View test runs	Not set

Clear explicit permissions

Save changes Undo changes

Permissions can be set both on specific users and on Team Foundation Server (TFS) user groups. If you are using Release Management in TFS, you can also use Active Directory groups to manage permissions. If you are using Release Management in Team Services, and your Team Services account is integrated with Azure Active Directory (AAD), you can also use AAD groups.

To set or over-ride the permissions for all release definitions in a project, open the shortcut menu from the ▼ icon next to **All release definitions**. To set or over-ride the permissions for a specific release definition, open the shortcut menu from the ▼ icon next to that definition name. Then choose **Security** to open the PERMISSIONS dialog.



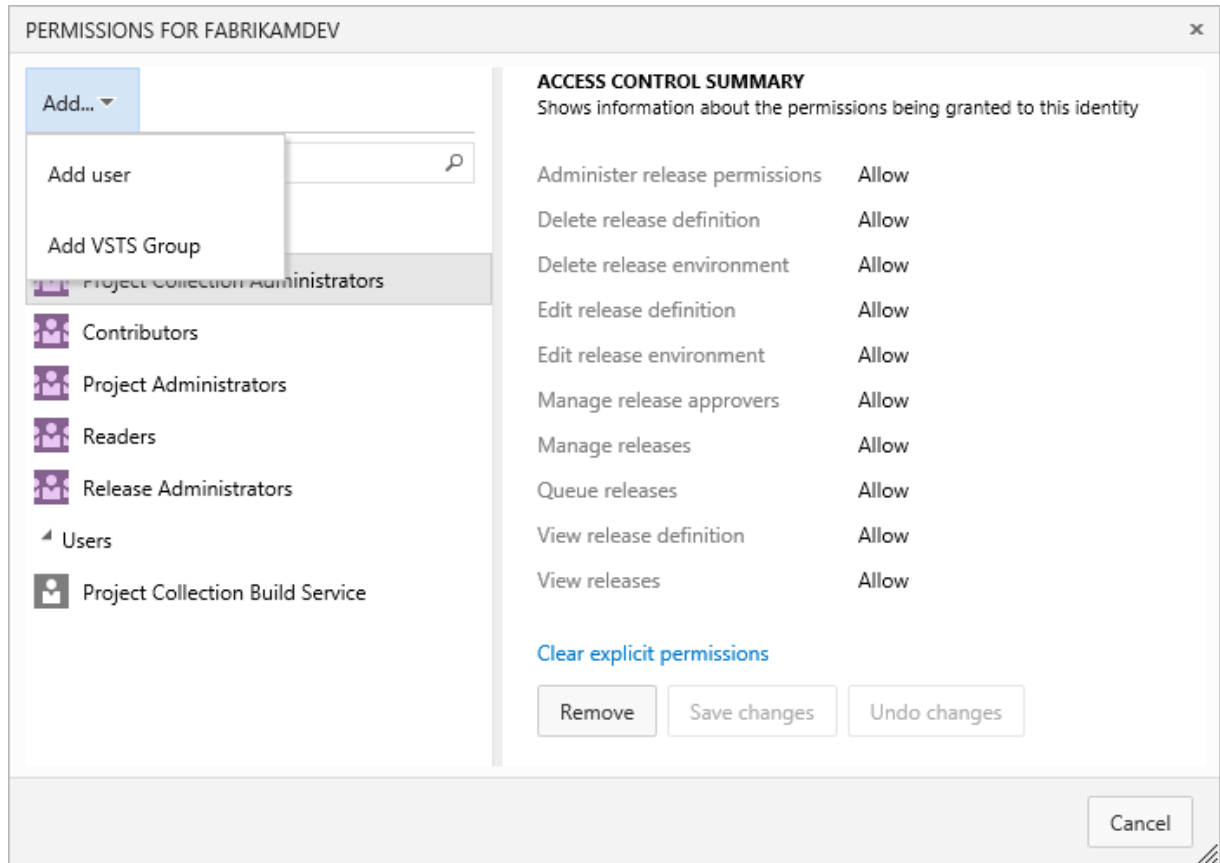
The following permissions are defined in Release Management.

Permission	Description	Scopes
<b>Administer release permissions</b>	Can change any of the other permissions listed here.	Project, Release definition, Environment
<b>Edit release definition</b>	Can save any changes to a release definition, including configuration variables, triggers, artifacts, and retention policy as well as configuration within an environment of the release definition. To make changes to a specific environment in a release definition, the user also needs <b>Edit release environment</b> permission.	Project, Release definition
<b>Edit release environment</b>	Can edit environment(s) in release definition(s). To save the changes to the release definition, the user also needs <b>Edit release definition</b> permission. This permission also controls whether a user can edit the configuration inside the environment of a specific release instance. The user also needs <b>Manage releases</b> permission to save the modified release.	Project, Release definition, Environment
<b>Manage releases</b>	Can edit the configuration in releases. To edit the configuration of a specific environment in a release instance, the user also needs <b>Edit release environment</b> permission.	Project, Release definition
<b>Delete release definition</b>	Can delete release definition(s).	Project, Release definition

Permission	Description	Scopes
<b>Delete release environment</b>	Can delete environment(s) in release definition(s).	Project, Release definition, Environment
<b>Delete releases</b>	Can delete releases for a definition.	Project, Release definition
<b>Manage release approvers</b>	Can add or edit approvers for environment(s) in release definition(s). This permissions also controls whether a user can edit the approvers inside the environment of a specific release instance.	Project, Release definition, Environment
<b>Queue releases</b>	Can create new releases.	Project, Release definition
<b>Manage deployments</b>	Can initiate a direct deployment of a release to an environment. This permission is only for direct deployments that are manually initiated by selecting the <b>Deploy</b> action in a release. If the condition on an environment is set to any type of automatic deployment, the system automatically initiates deployment without checking the permission of the user that created the release.	Project, Release definition, Environment
<b>View release definition</b>	Can view release definition(s).	Project, Release definition
<b>View releases</b>	Can view releases belonging to release definition(s).	Project, Release definition

Default values for all of these permissions are set for select team project collection and team project groups. For example, **Project Collection Administrators**, **Project Administrators**, and **Release Administrators** are given all of the above permissions by default. **Project Contributors** are given all permissions except **Administer release permissions**. **Project Readers**, by default, are denied all permissions except **View Release Definitions** and **View Releases**.

When setting global permissions for all release definitions, add users and Team Services groups by using the **Add** menu. Use the **Clear explicit permissions** link to allow project-level settings for groups or users to be inherited from the main collection settings. Use the **Remove** button to remove groups or users.



When you set permissions for a specific release definition, you can choose to inherit and over-ride the global permissions for just this release definition. Notice the useful **Why?** link that appears as you hover over a permission. Use it to trace the inheritance of that permission for the selected release definition.



PERMISSIONS FOR FABRIKAMDEV

Add... ▾
Inheritance ▾

✓ On
Off

Search
VSTS Gro
Project Collection Administrators
Contributors
Project Administrators
Readers
Release Administrators
Users
Project Collection Build Service

### ACCESS CONTROL SUMMARY

Shows information about the permissions being granted to this identity

Administer release permissions	Inherited allow	
Delete release definition	Inherited allow	<a href="#">Why?</a>
Delete release environment	Inherited allow	
Edit release definition	Inherited allow	
Edit release environment	Inherited allow	
Manage release approvers	Inherited allow	
Manage releases	Inherited allow	
Queue releases	Inherited allow	
View release definition	Inherited allow	
View releases	Inherited allow	

Clear explicit permissions

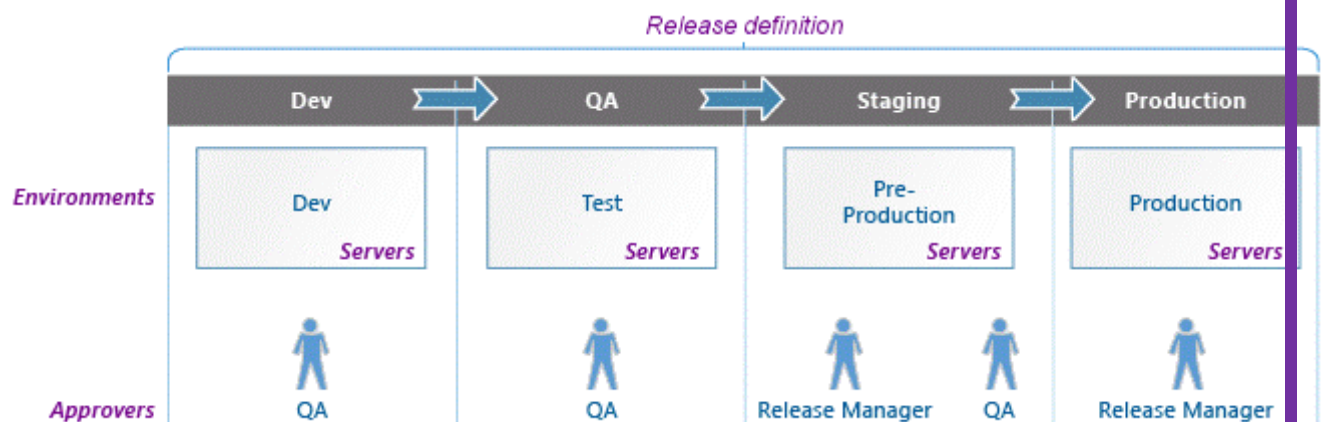
Remove
Save changes
Undo changes

Cancel

If you turn off inheritance in this dialog, you can set explicit permissions for this release definition.

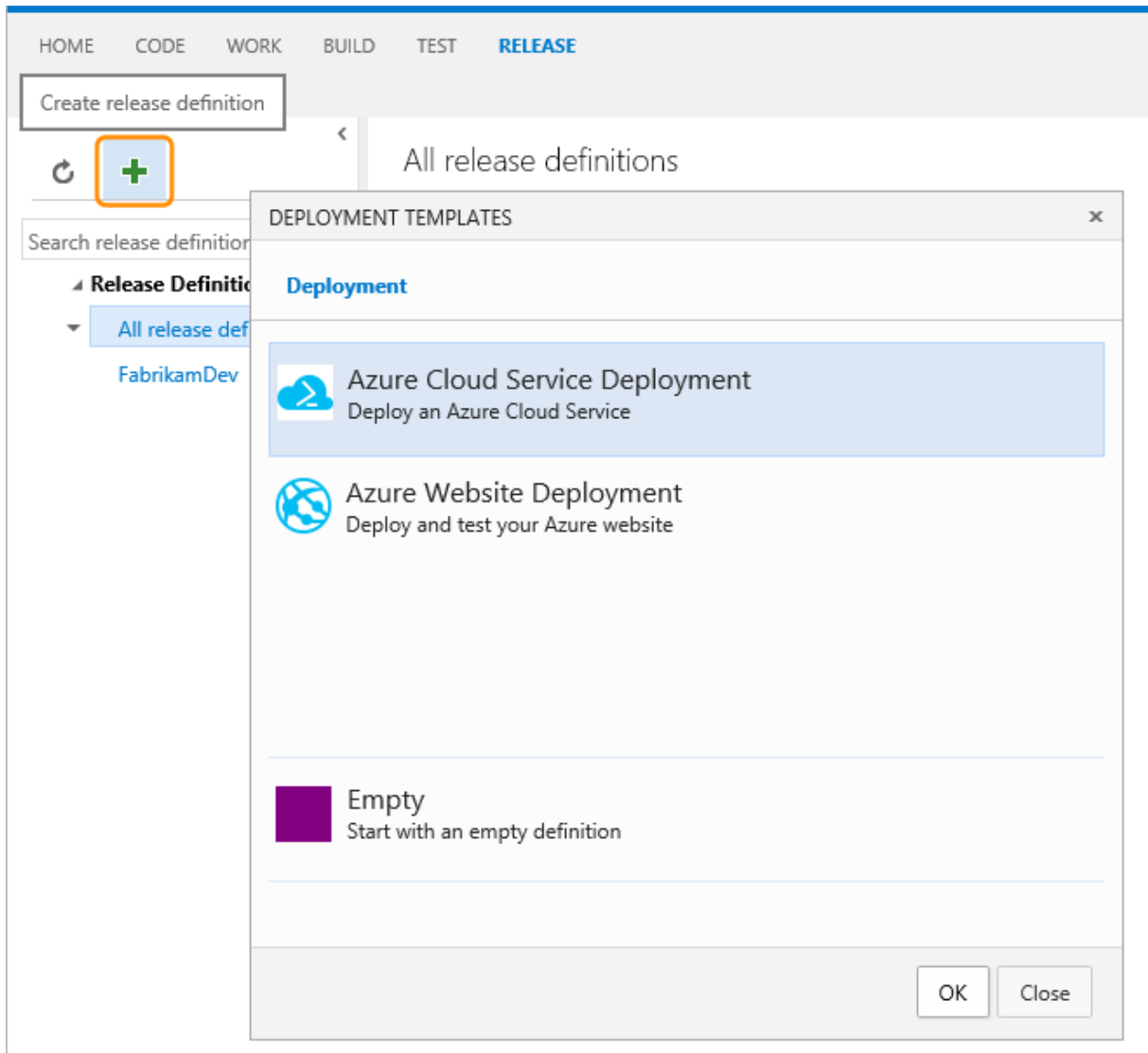
## 6.2.2 Add environments

A release definition is a collection of environments. An **environment** is a *logical* entity that represents where you want to deploy a release. Physically, the deployment in an environment may happen to a collection of servers, a cloud, multiple clouds, or an app store. The deployment steps in an environment are described using tasks.



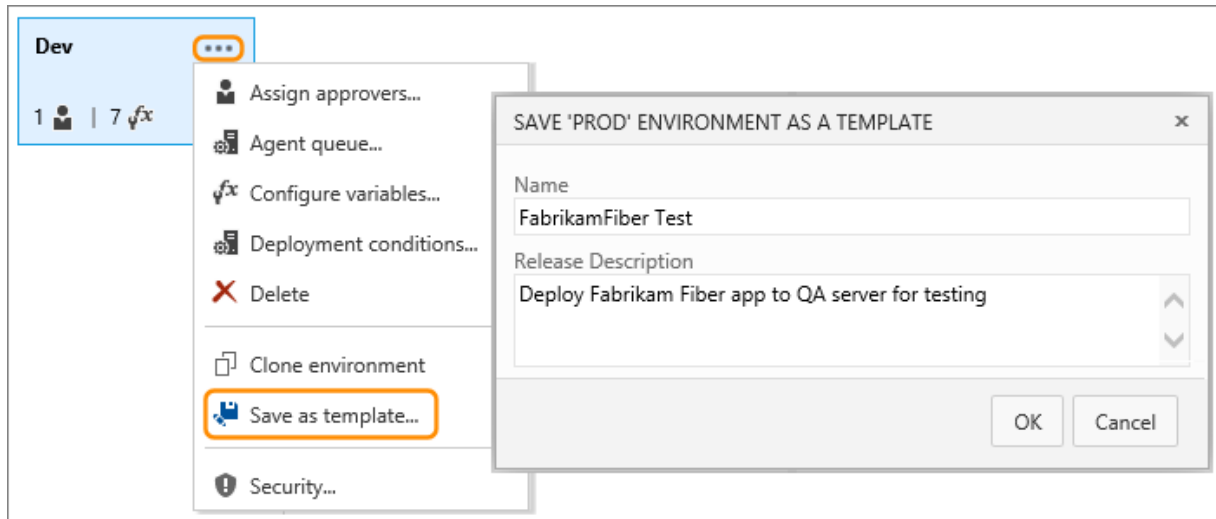
### 6.2.2.1 Environment templates

When you start a new release definition, or when you add an environment to an existing release definition, you can choose from a list of deployment templates for each environment.

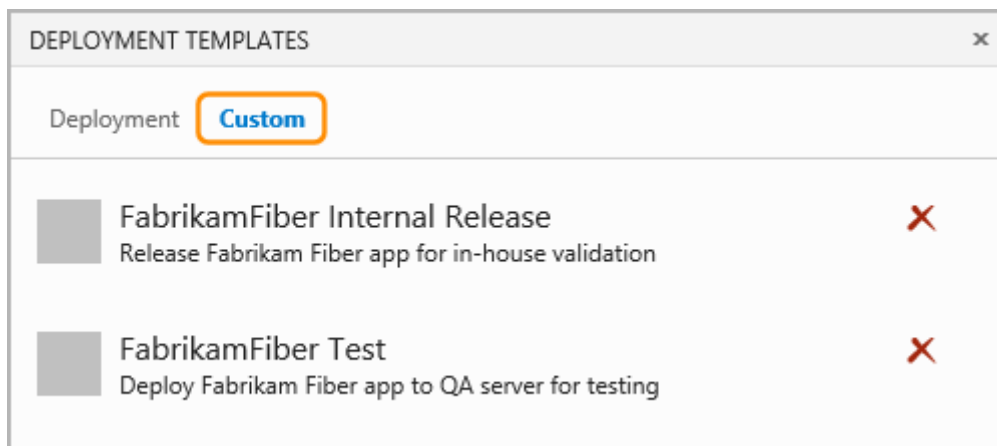


These templates pre-populate the environment with the appropriate tasks and settings, which can considerably reduce the time and effort required to create a release definition. Alternatively, you can choose to start with an empty release definition that contains only a single default empty environment, or an empty environment that contains no tasks.

You can also create your own custom environment templates from an environment you have populated and configured. Choose **Save as template** on the shortcut menu that opens from the environment's ellipses (...). Enter a name and description, and choose **OK**.



The custom templates you create appear in the **Custom** tab of the deployment templates dialog.

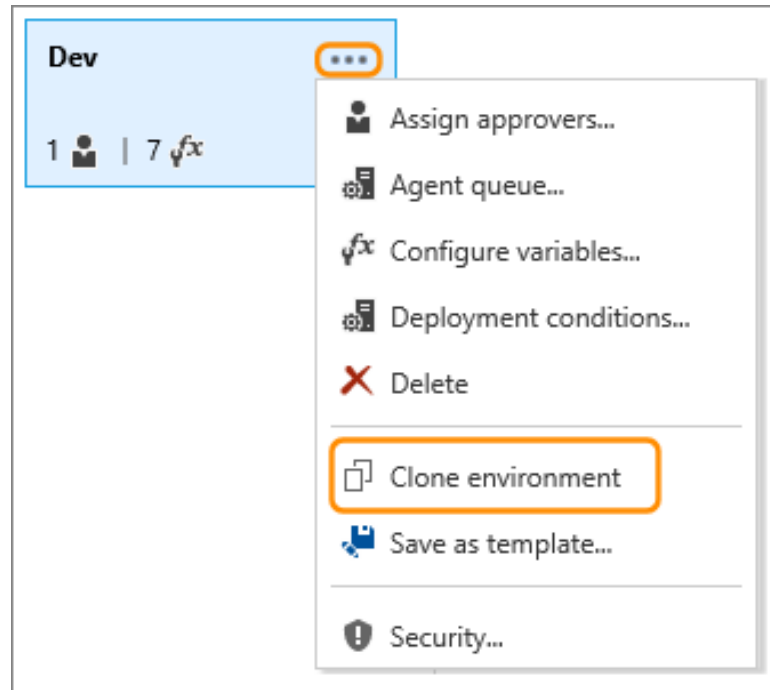


#### 6.2.2.2 Closing environments

A release definition often contains several environments such as development, testing, QA, and production. Typically, all of these environments are fundamentally similar - and the techniques used to set up and deploy to each one are the same with the exception of minor differences in configuration for each environment and task (such as target URLs, service paths, and server names).

After you have added an environment to a release definition and configured it by adding tasks and setting the properties for each one, you can clone it to create another environment within the same definition. The cloned environment has the same tasks, task properties, and configuration settings.

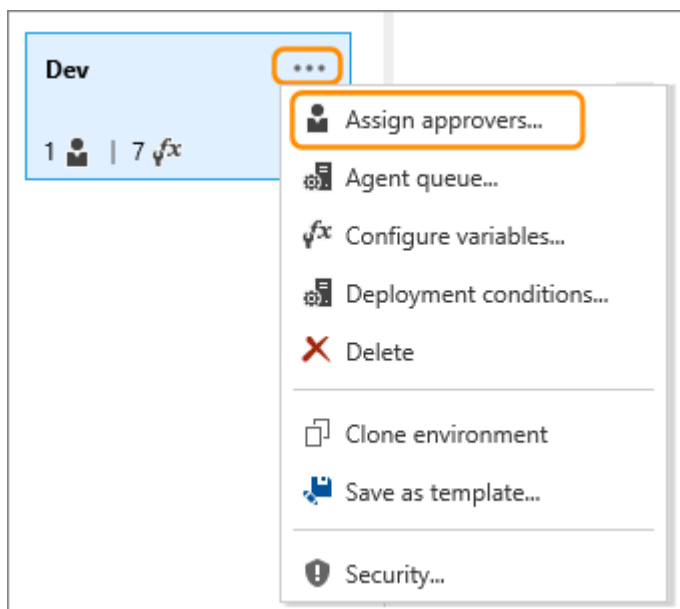
To create a copy of an environment within a release definition, choose **Clone environment** on the shortcut menu that opens from the environment's ellipses (...) and enter a name for the new environment.



#### 6.2.2.3 Approvals and approvers

You can define approvers for each environment in a release definition. When a release is created from a release definition that contains approvers, the deployment stops at each point where approval is required until the specified approver grants approval or rejects the release (or re-assigns the approval to another user).

Use the ellipses (...) to show the environment pop-up menu and choose **Assign approvers**.



When the **Automated** checkbox in the **Approvals** dialog is set for an approval step, the deployment is automatically approved. Notice that you can edit the environment owners list in this dialog. Environment owners are notified whenever the deployment is completed.

CONFIGURE - 'DEV' ENVIRONMENT

ApprovalsQueueVariablesGeneralDeployment conditions

Approvers

Select the users who can approve or reject the deployments to this environment. ⓘ

Pre-deployment approver

☒ Automatic

☐ Specific Users

Post-deployment approver

☒ Automatic

☐ Specific Users

Options

Apply these rules for the deployment approvers

☐ Send an email notification to the approver whom the approval is pending on

☐ The user who creates the release cannot approve

OKCancel

To specify one or more approvers for an operation, select the **Specific Users** option and enter each approver's email address. You can enter part of the name and choose from the drop-down list of matching user names defined for the project.

CONFIGURE - 'DEV' ENVIRONMENT


Approvals
Queue
Variables
General
Deployment conditions

### Approvers

Select the users who can approve or reject the deployments to this environment. ⓘ

Pre-deployment approver

☐ Automatic
☒ Specific Users



Raisa Pokrovska
✕

Showing 1 results

Post-deployment approver

☒ Showing 1 results
☐ Specific Users

### Options

Apply these rules for the deployment approvers

☐ Send an email notification to the approver whom the approval is pending on
☐ The user who creates the release cannot approve

OK

Cancel

You can add multiple approvers for both pre-deployment and post-deployment settings.

CONFIGURE - 'DEV' ENVIRONMENT

Approvals
Queue
Variables
General
Deployment conditions

### Approvers

Select the users who can approve or reject the deployments to this environment. ⓘ

Pre-deployment approver

☐ Automatic
☒ Specific Users

Search users and groups

☒ All of the specified users in any order
☐ Any one of the specified users
☐ All of the specified users in sequential order

[Fewer Options](#)

When you add multiple approvers, you can also control how they can approve the deployment:

- **All of the specified users in any order:** This is useful if you want sign-off from a set of users, all of them have to approve, and it does not matter in what order they approve.
- **Any one of the specified users:** This is useful if you want sign-off from only one of a set of users. As long as one of the users approves, the release moves forward.
- **All of the specified users in sequential order:** This is useful if you want sign-off from a set of users, all of them have to approve, and you want them to approve in the specified order. For example, the second user can approve only after the first user approves, and so on.

You can also specify account (user) groups as approvers. When a group is specified as an approver, only one of the users in that group needs to approve in order for the release to move forward.

- If you are using **Visual Studio Team Services**, you can use local groups managed in Team Services or Azure Active Directory (AAD) groups if they have been added into Team Services.
- If you are using **Team Foundation Server (TFS)**, you can use local groups managed in TFS or Active Directory (AD) groups if they have been added into TFS.

CONFIGURE - 'DEV' ENVIRONMENT

**Approvals** Queue Variables General Deployment conditions

**Approvers**

Select the users who can approve or reject the deployments to this environment. ⓘ

Pre-deployment approver ☐ Automatic ☒ Specific Users

[Fabrikam]\Fabrikam Team X Search users and groups

You can also use the options in the **Approvals** dialog to:

- Specify whether the approvers will receive an email when an approval is awaiting their attention.
- Prevent the user who created a release from approving his or her own release. This is often useful to ensure compliance with corporate audit requirements.

CONFIGURE - 'DEV' ENVIRONMENT

Approvals
Queue
Variables
General
Deployment conditions

### Approvers

Select the users who can approve or reject the deployments to this environment. ⓘ

Pre-deployment approver

☐ Automatic
☒ Specific Users

[Fabrikam]\Fabrikam Team X Search users and groups

More Options

Post-deployment approver

☒ Automatic
☐ Specific Users

### Options

Apply these rules for the deployment approvers

☒ Send an email notification to the approver whom the approval is pending on
☒ The user who creates the release cannot approve

OK

Cancel

#### 6.2.2.4 Environment deployment triggers

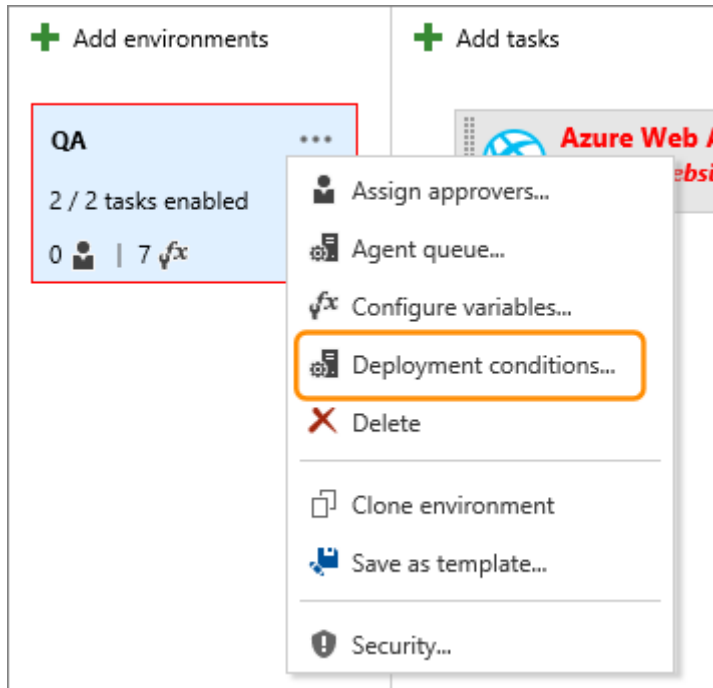
You can configure a **continuous deployment trigger** in a release definition so that a new release is created automatically whenever a new version of a linked artifact is available. However, you must also configure *deployment triggers* for each environment in the definition in order for the new release to be deployed automatically to one or more of these environments.

Deployment triggers are configured for each environment, but the combination of these allows you to orchestrate the overall deployment - such as the sequence in which automated deployments occur across all the environments in a release definition. For example, you can set up a linear pipeline where a release is deployed first to the **Dev** environment, then to the **QA** environment and finally, if the deployment to **QA** succeeds, to the **Prod** environment.

Alternatively, you can set up a release process where a build is deployed to a number of **QA** environments in parallel as soon as it completes, but the release to the **Prod** environment must always be deployed manually by selecting a successful release and promoting it.

To configure the triggers for an environment, choose **Deployment conditions** on the shortcut menu that opens from the environment's ellipses (...).





Select the appropriate option in the **Trigger** section.

CONFIGURE - 'DEV' ENVIRONMENT

Approvals Queue Variables General **Deployment conditions**

**Trigger**

Define when the deployment on this environment should be started automatically. If you want to deploy manually to this environment, use the "Deploy" action.

☐ No automated deployment, manual only

☒ After release creation

☐ After successful deployment

**Options**

Define behavior when multiple releases are waiting to be deployed on this environment. ⓘ

☒ Allow multiple releases to be deployed at the same time

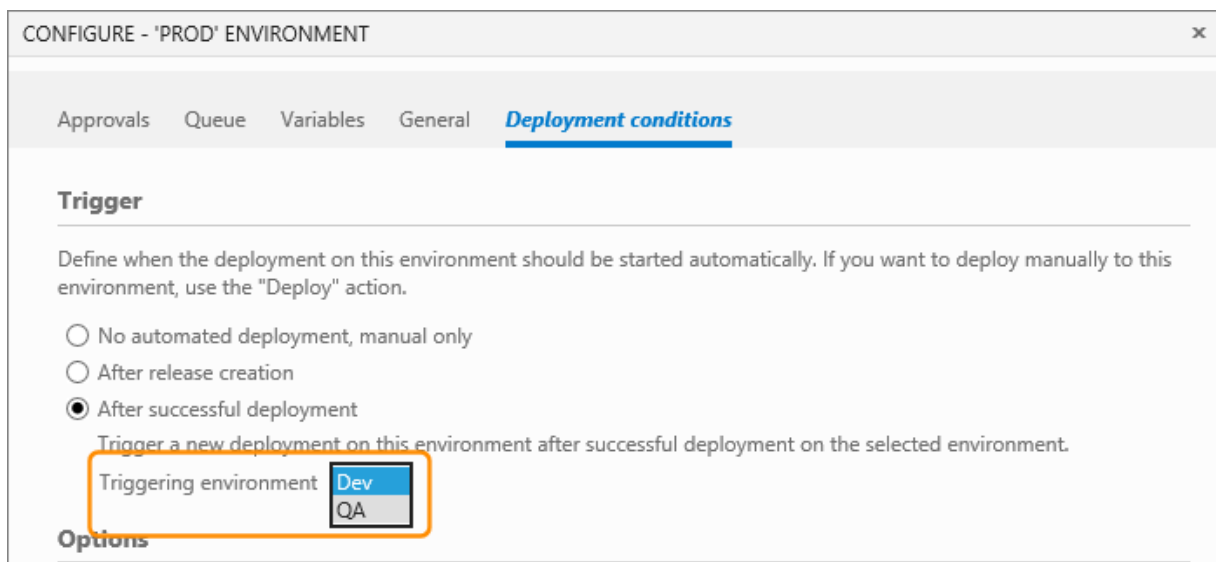
☐ Allow only one active deployment at a time

OK Cancel

The **Trigger** settings are:

- **No automated deployment, manual only:** Releases are not automatically deployed to this environment. To deploy a release to this environment, you must manually select the **Deploy** action when a new release is created. This is the default setting.

- **After release creation:** The release will be deployed to this environment as soon as it is created, due either to a user action (such as manually starting a new release from the UI) or because a **continuous deployment trigger** is configured for this release definition. For example, if you have checked the continuous deployment trigger option for a release definition and selected a specific build artifact, the deployment to this environment will start automatically when the output from a new build is available.
- **After successful deployment:** Use this setting if you want the release to be first deployed and validated in another environment before it is deployed to this environment. For example, you can set up a linear pipeline of deployment through **Dev**, **QA**, and **Prod** environments by setting the trigger on the **Dev** environment to **After release creation**, the trigger on the **QA** environment to **After successful deployment**, and the trigger on the **Prod** environment to **Dev**, and the trigger on Prod environment to **After successful deployment** to the **QA** environment. You must have at least two environments configured in the release definition to enable this option, and - when you select it - a list of the other environments is shown where you can select the **Triggering environment**.



CONFIGURE - 'PROD' ENVIRONMENT

Approvals Queue Variables General Deployment conditions

**Trigger**

Define when the deployment on this environment should be started automatically. If you want to deploy manually to this environment, use the "Deploy" action.

☐ No automated deployment, manual only  
☐ After release creation  
☒ After successful deployment

Trigger a new deployment on this environment after successful deployment on the selected environment.

Triggering environment Dev  
QA

**Options**



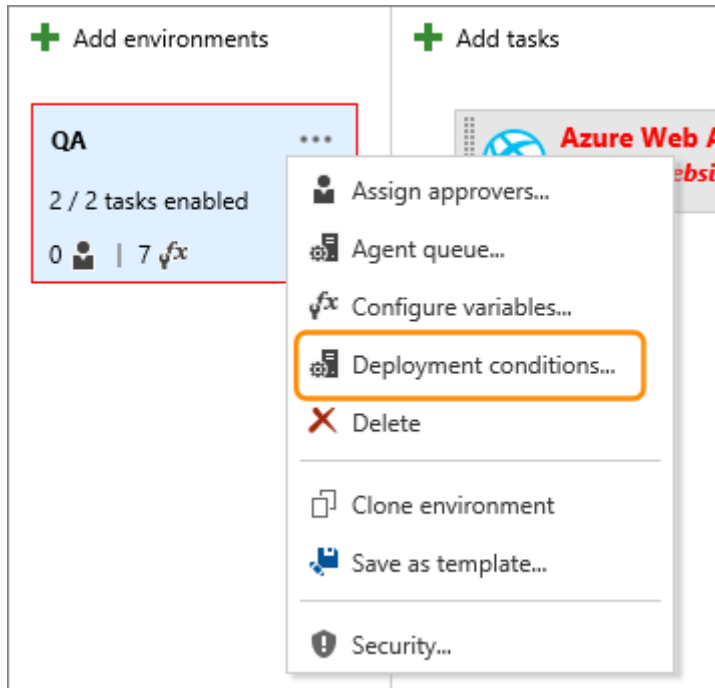
Note that you can always deploy a release directly to any of the environments in your release definition by selecting the **Deploy** action when you create a new release. In this case, the deployment conditions you configure, such as a trigger on successful deployment to another environment, do not apply. The deployment occurs irrespective of these settings. This gives you the ability to override the release process. Performing such direct deployments requires the **Manage deployments** permission, which should only be given to selected and approved users.

Users with permission to edit release definitions can also configure environment deployment triggers in the **Triggers tab** of a release definition.

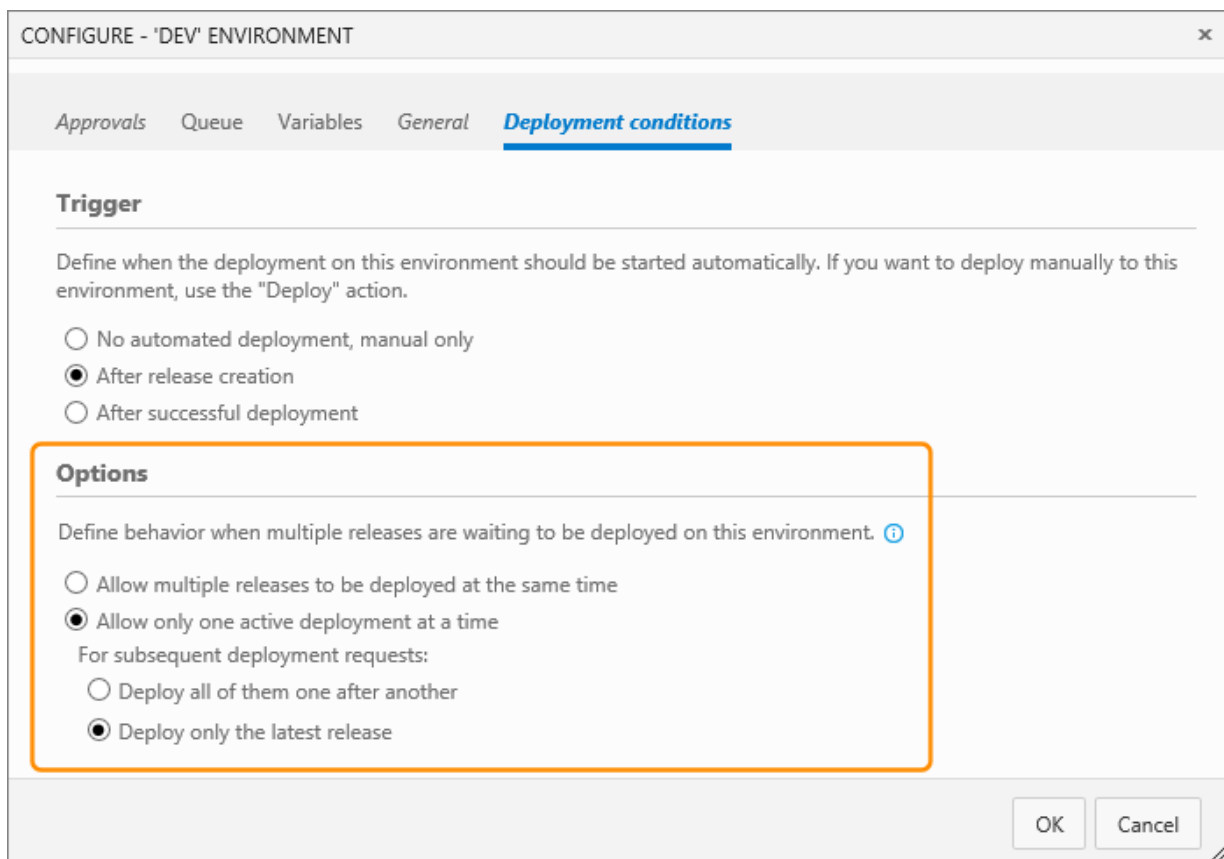
#### 6.2.2.5 Queuing policies

In some cases, you may be generating builds more quickly than they can be deployed. Alternatively, multiple users may be creating releases from the same release definition for deployment of different artifacts. In such cases it's useful to be able to control how multiple releases are queued into an environment, and *queuing policies* give you that control.

To configure the queuing policy for an environment, choose **Deployment conditions** on the shortcut menu that opens from the environment's ellipses (...).



Queuing policies are defined in the **Options** section of the **Deployment conditions** tab.



The options you can choose for a queuing policy are:

- **Allow multiple releases to be deployed at the same time:** Use this option if you dynamically provision new resources in your environment, and it is physically capable of handling the deployment of

multiple releases in parallel. Because the artifacts are deployed to independent physical resources, there is no conflict for these resources. This is typically the case in test environments.

- **Allow only one active deployment at a time:** If you select this, two more options appear:
  - **Deploy all of them one after another:** Use this option if you want to deploy all the releases sequentially into the same shared physical resources. By deploying them in turn, one after the other, you ensure that two deployment jobs do not target the same physical resources concurrently, even if there are multiple build and deployment agents available. You also ensure that pre-deployment approval requests for the environment are sent out in sequence.
  - **Deploy only the latest release:** Use this option if you are producing releases faster than builds, and you only want to deploy the latest build.

To understand how these options work, consider a scenario where releases **R1**, **R2**, ..., **R5** of a single release definition are created in quick succession due to new builds being produced rapidly. Assume that the first environment in this definition is named **QA** and has both pre-deployment and post-deployment approvers defined.

- If you select **Allow multiple releases to be deployed at the same time**, all five approval requests will be sent out as soon as the releases are created. If the approvers grant approval for all of the releases, they will all be deployed to the **QA** environment in parallel. (if the **QA** environment did not have any pre-deployment approvers defined, all the five releases will automatically be deployed in parallel to this environment).
- If you select **Allow only one active deployment at a time** and **Deploy all of them one after another**, the pre-deployment approval for release **R1** will be sent out first. After this approval is completed, the deployment of release **R1** to the **QA** environment begins. Next, a request for post-deployment approval is sent out for release **R1**. It is only after this post-deployment approval is completed that execution of release **R2** begins and its pre-deployment approval is sent out. The process continues like this for all of the releases in turn.
- The only other option, **Allow only one active deployment at a time** and **Deploy only the latest release**, is similar to the previous option. However, releases **R2**, **R3**, and **R4** will be skipped, and the pre-deployment approval for **R5** in the **QA** environment will be sent out immediately after the post-deployment approval for release **R1** is completed.

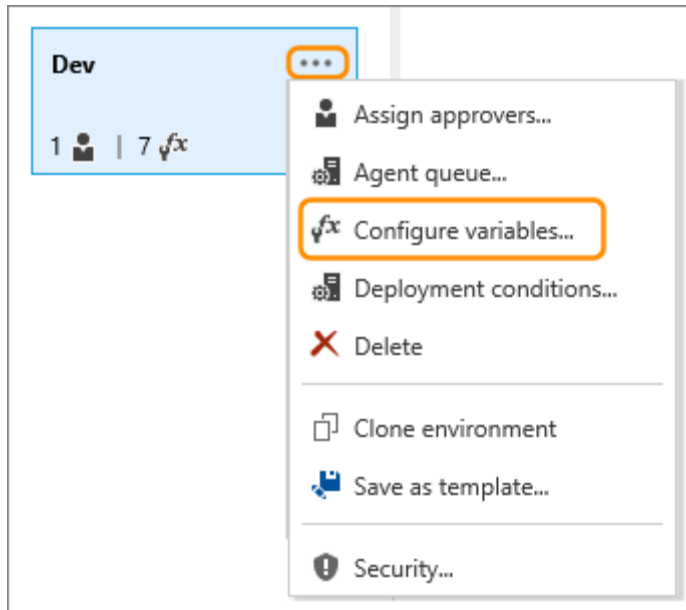
Users with permission to edit release definitions can also configure the deployment queuing policies in the **Triggers tab** of a release definition.

#### 6.2.2.6 Configuration properties

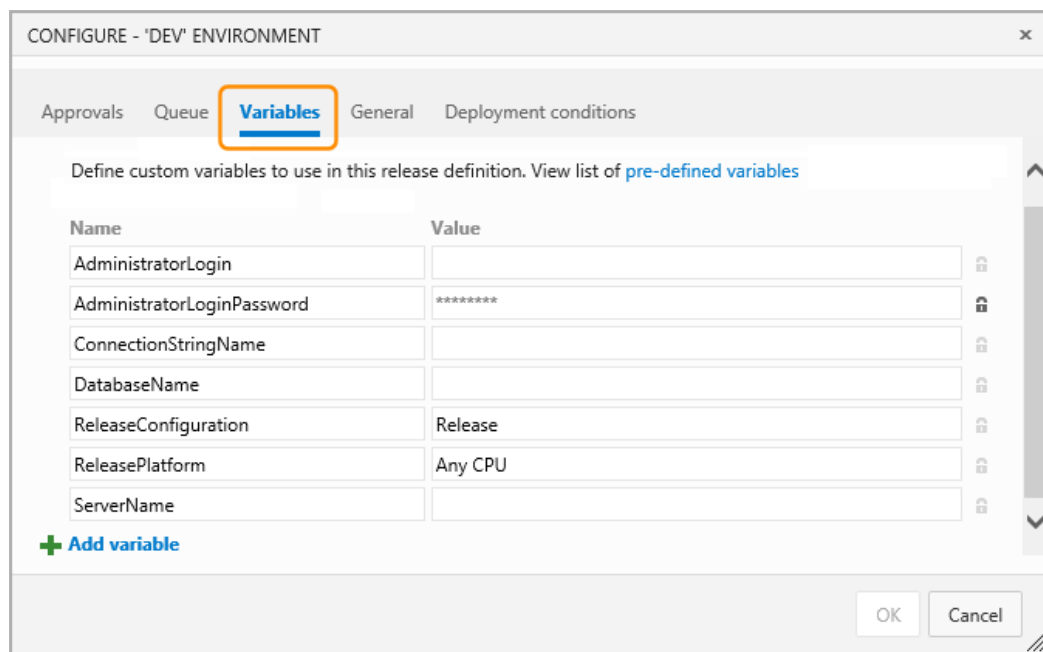
You can configure properties for environments by defining custom variables. Both a release definition and an individual environment can define configuration properties that are used in environments and tasks. You can:

- Share values across all of the environments. Share values across all of the tasks within one specific environment.
- Avoid duplication of values, making it easier to update all occurrences as one operation.
- Store sensitive values in a way that they cannot be seen or changed by users of the release definitions.

You can see and edit all of the variables defined for an individual environment within a release definition in the **CONFIGURE** dialog for the environment. Open this dialog by choosing **Configure variables** on the menu that opens from the environment's (...) ellipses.



Then select the **Variables** tab in the CONFIGURE dialog. You can add your own variables here, and hide sensitive values from users of the release definition by choosing the "padlock" icon. The values of hidden properties are stored securely by the service and these values cannot be viewed by users after they are saved. During a deployment, the Release Management service decrypts those values that are referenced by the tasks, and passes them to the agent over secure HTTPS channel.



You can also view and edit the configuration for each environment in the **Configuration** tab of a release definition by choosing **Environments** in the **View** list.

Definition: FabrikamDev | Releases

Environments
Artifacts
Configuration
Triggers
General

Release variables

Refresh
Save
+ Release

Environment variables

Release variables

View and modify custom variables defined on the environments in this release definition.

	Dev	Test	Production
AdministratorLogin			
AdministratorLoginPassword	*****	*****	*****
ConnectionStringName			
DatabaseName			
ReleaseConfiguration	Release	Release	Release
ReleasePlatform	Any CPU	Any CPU	Any CPU

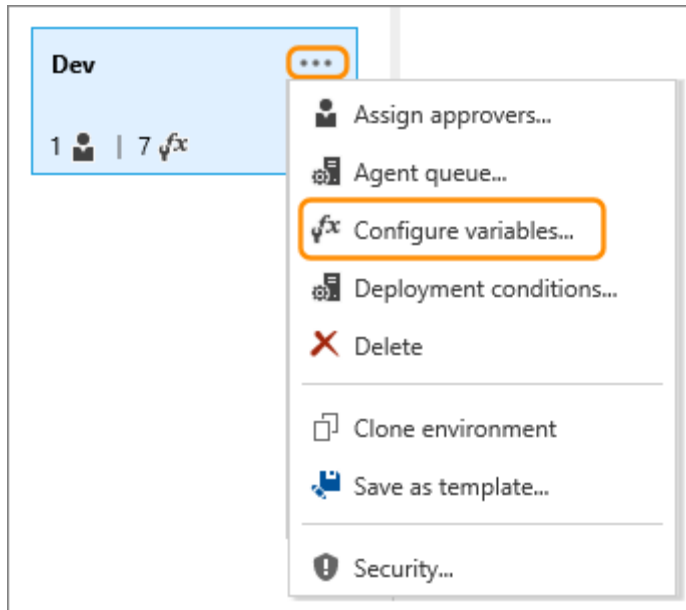
If you have very many environments, you can simplify the view by selecting them in the drop-down list of environments next to the **View** list.

Environment variables

☒ Dev
☒ Test
☒ Production

#### 6.2.2.7 General options

While the most important part of defining an environment is the automation tasks, an environment also has options that you can configure. Open the **CONFIGURE** dialog by choosing **Configure variables** on the menu that opens from the environment's (...) ellipses.



Then select the **General** tab. The following options are available to control the deployment:

- **Environment owner:** You can designate a single user or a single group to be the environment owner.
- **Send email notifications:** The environment owner can receive email notifications when deployments occur to this environment. This can be for every deployment, or just for failed deployments.
- **Skip artifacts download:** You may choose to skip the **download of artifacts** for a specific environment. Use this option if you want to implement your own custom logic for downloading artifacts by using tasks.
- **Deployment job timeout in minutes:** Use this option if you want to specify the job timeout for this environment. A zero value for this option means that the timeout is effectively infinite and so, by default, jobs run until they complete or fail.

CONFIGURE - 'DEV' ENVIRONMENT

Approvals

Queue

Variables

General

Deployment conditions

Environment owner

Raisa Pokrovskaya

Send email notifications

☐ Always
☒ Only on failure
☐ Never

Skip artifacts download

☐

Deployment timeout in minutes

30

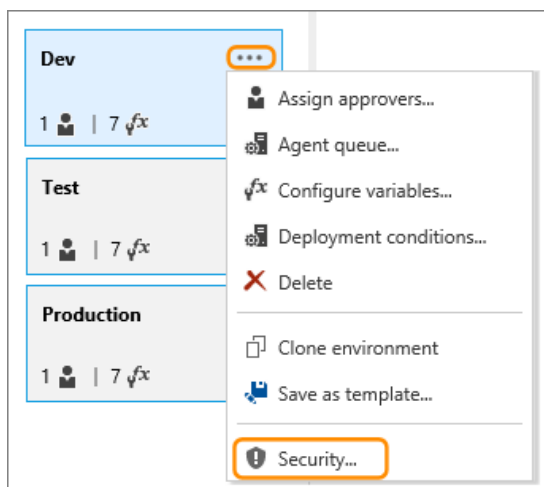
OK

Cancel

#### 6.2.2.8 Environment security

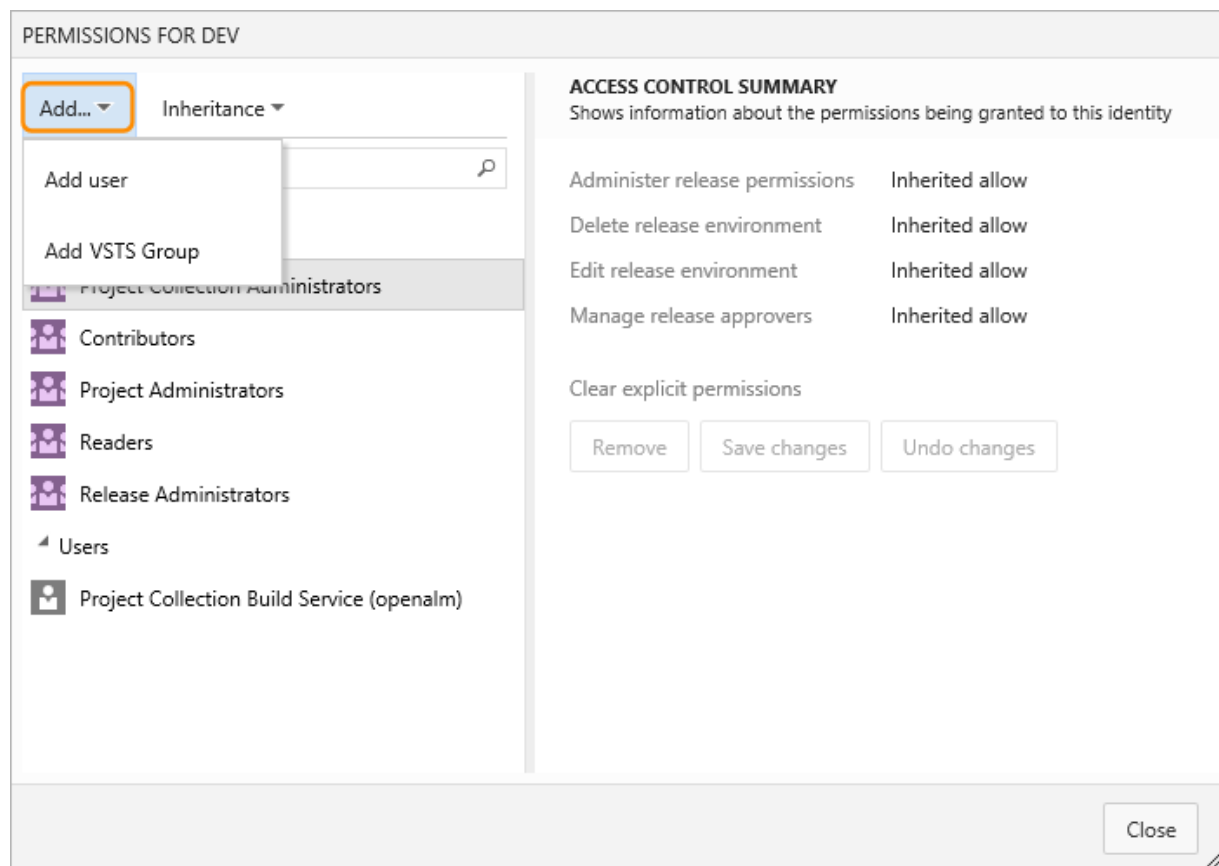
Permissions in Release Management define the authorizations that can be granted or denied to users and groups. These permissions can be granted or denied in a hierarchical model at the team project level, for a specific release definition, or for a specific environment in a release definition. Within this hierarchy, permissions can be inherited from the parent or overridden. To understand the permissions available, and how these are set at project and release definition level, see **Security** in the release definitions topic.

To specify security settings for individual environments in a release definition, open the **PERMISSIONS** dialog by choosing **Security** on the shortcut menu that opens from the ellipses (...) on an environment.

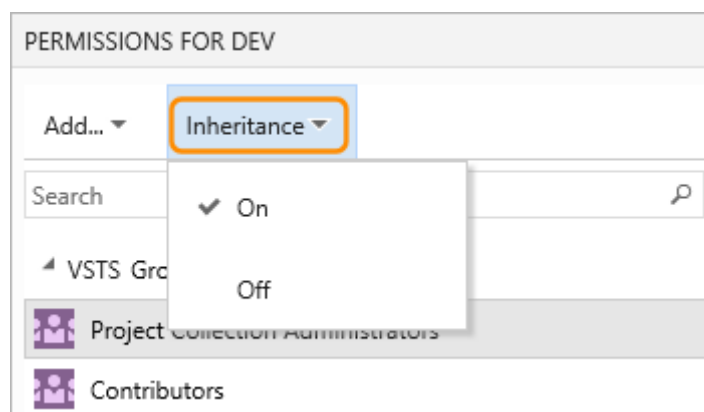




When setting permissions for an environment, add users and Team Services groups by using the **Add** menu. Use the **Clear explicit permissions** link to allow settings for groups or users to be inherited from parent project and release definitions. Use the **Remove** button to remove groups or users.



When you set permissions for an environment, you can choose to inherit and over-ride the permissions from the containing release definition.

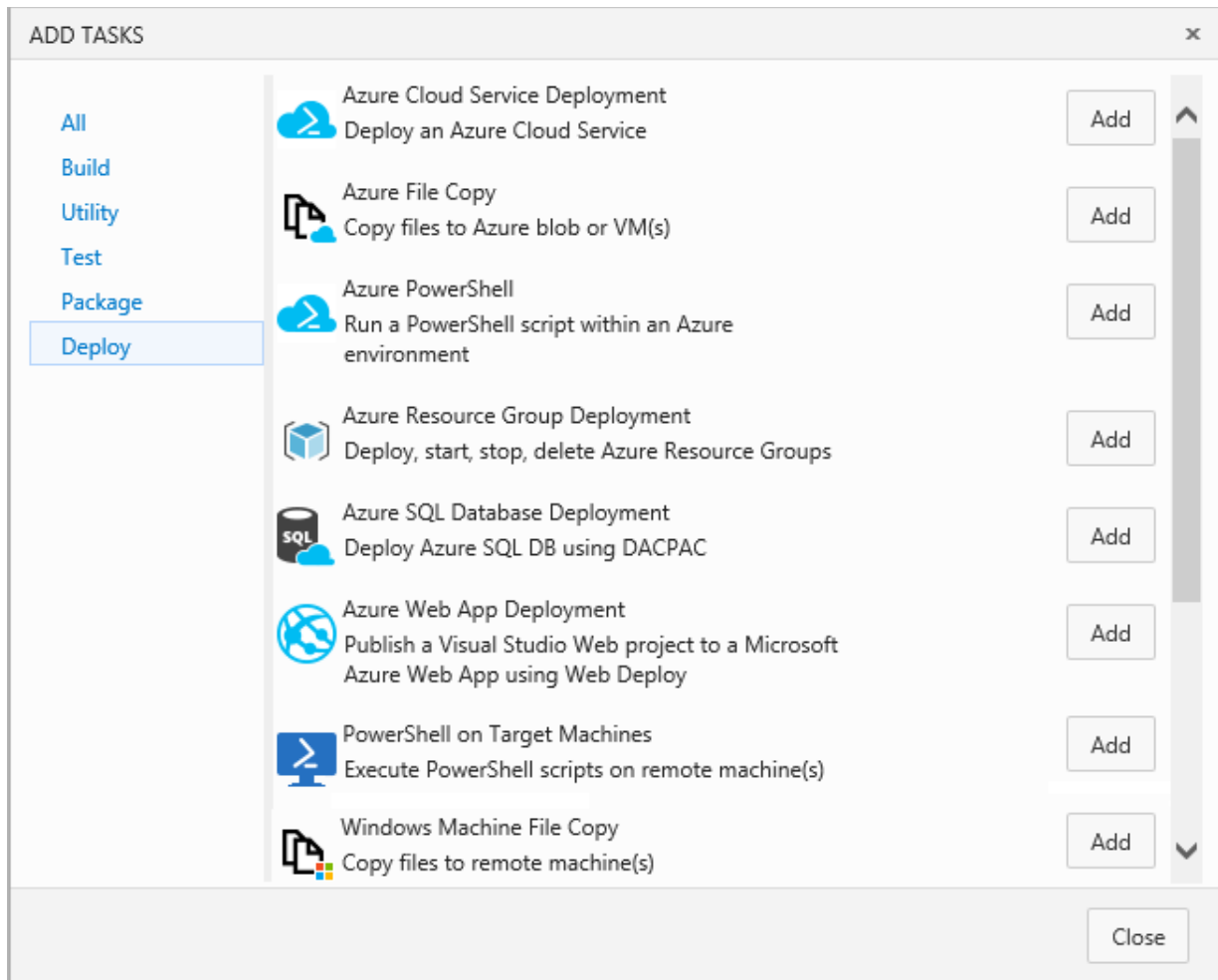


### 6.2.3 Add tasks

A release definition is a collection of environments, each of which contains one or more tasks. A **task** is an automation object that performs the mechanics of a deployment, or an action related to a deployment. For example, a task may prepare an environment, copy files, run tests, or perform clean up.

### 6.2.3.1 Out-of-the-box tasks

To see a list of the currently installed tasks, create a release definition and add an environment. Choose the plus icon (+) to add a task. The task selector shows tasks that are suitable for use in a release definition.



Each task has a **More information** link at the bottom of the parameters pane.

**Deploy Website to Azure**

Azure Subscription: AzureConnection1 [Manage](#)

Web App Name: FabrikamDev

Web App Location: South Central US

Slot:

Web Deploy Package: \$(Agent.ReleaseDirectory)\\*\*\\*.zip

Additional Arguments:

**Control Options**

Enabled ☒

Continue on error ☐

Always run ☐

[More Information](#)

Here are more details about some of the standard tasks available in the **Deploy** section of the **ADD TASKS** dialog in a release definition. Additional tasks can be obtained from the Visual Studio Marketplace.

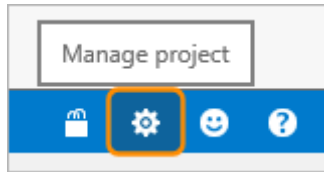
Task name	Description
Azure Cloud Service Deployment	Deploys the application as a package to an Azure Cloud Services (Web and Worker role) website. The build step must create an Azure Cloud Services deployment package and corresponding CSCFG configuration file. The Azure subscription must already contain a storage account. The task creates a new Cloud Services (web and worker roles) instance using the name in the configuration file if it does not already exist.
Azure File Copy	Copies application files and other artifacts that are required to install the application on an Azure VM. Copies files to Azure blob storage and then to VMs. Uses <b>AzCopy</b> , the command-line utility for fast copying of data from and into Azure storage accounts.
Azure PowerShell	Deploys the application using Windows Azure PowerShell commands in a script you provide. The script must contain all the commands required to configure the Azure target instance and deploy the application package created by the build step. Paths and other information can be passed to the script as arguments using the built-in properties such as <b>\$(Agent.ReleaseDirectory)</b> .
Azure Resource Group Deployment	Deploys an application and associated resources to an Azure Resources Group. The task creates a new Azure Resources Group with the specified name if it does not already exist. Can also start, stop, restart, and delete virtual machines in a Resource Group.
Azure SQL Database Deployment	Deploys an Azure SQL Database to an existing Azure SQL Database server using DACPAC and SqlPackage.exe. Provides fine-grained control over database creation and upgrades, including schema, triggers, stored procedures, roles, users, and extended properties.
Azure Web App Deployment	Deploys the application to an Azure <i>Web Apps in App Service</i> website. The build step must create a single file deployment package suitable for an Azure Web Apps in App Service instance. The task creates a new website with the specified name if it does not already exist.
PowerShell on Target Machines	Deploys the application by executing a Windows PowerShell script on the target server(s). Can execute both PowerShell and PowerShell-DSC scripts. The group of machines can be specified as a list or as a variable.
Windows Machine File Copy	Deploys artifacts required to install an application on a remote server. Uses RoboCopy, the command-line utility built for fast copying of data. The group of machines can be specified

#### 6.2.3.2 Service endpoints

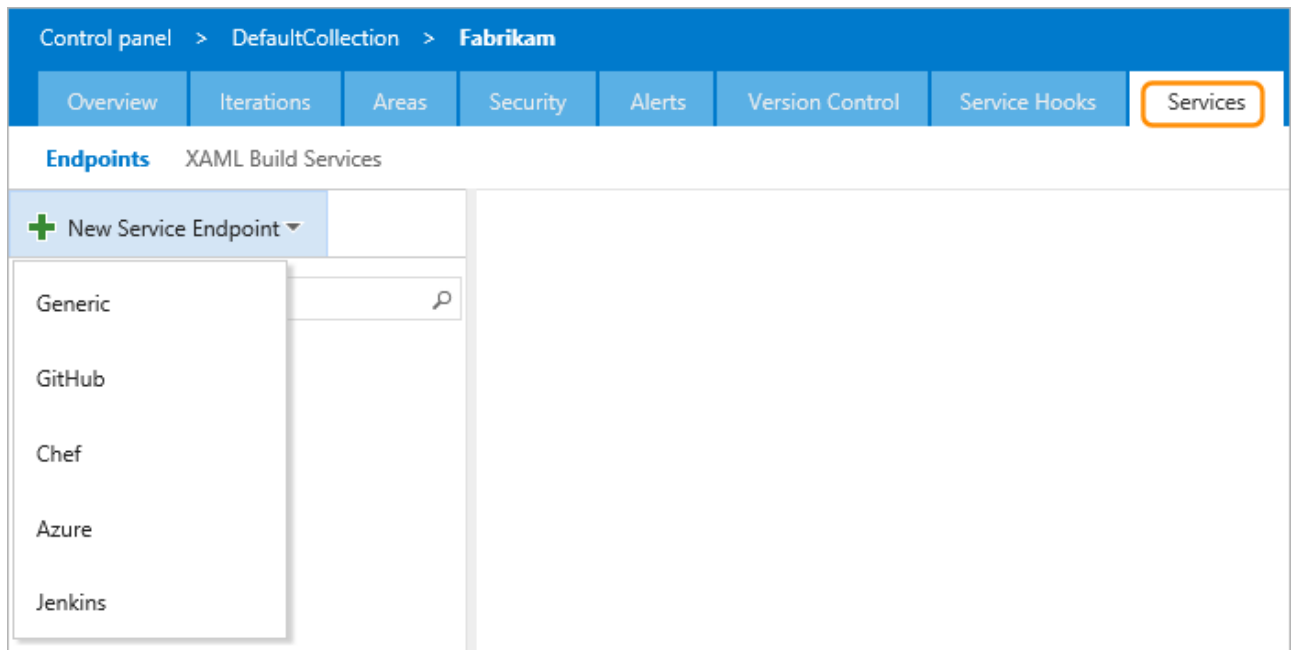
You will typically need to connect to external and remote services to execute tasks for a deployment. For example, you may need to connect to your Microsoft Azure subscription, to a different build server or file server, to an online continuous integration environment, or to services you install on remote computers.

You can define endpoints in Release Management that are available for use in all your tasks. For example, you can create an endpoint for your Azure subscription and use this endpoint name in an Azure Web Site Deployment task in a release definition.

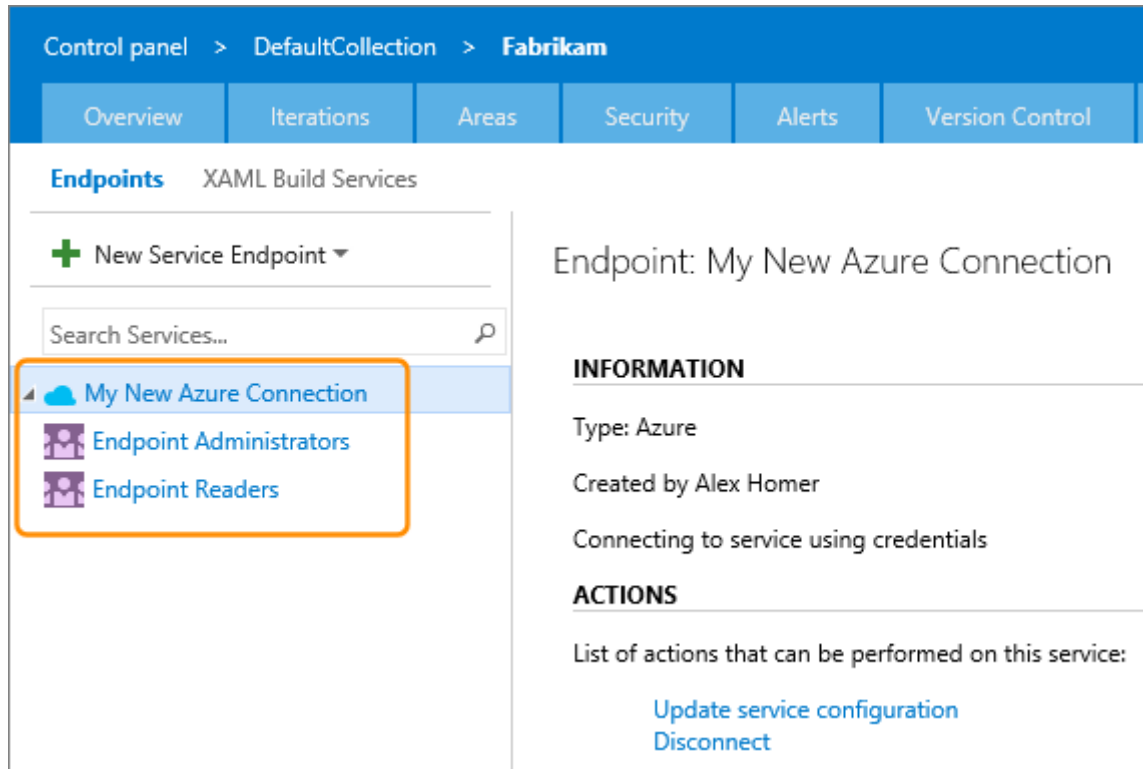
With your project open in Team Services, open the project Control Panel using the **Manage project** icon at the top right of the window.



Open the **Services** tab. Select the type of endpoint you want from the **New Service Endpoint** drop-down list.



Sensitive values you enter for an endpoint, such as the credentials, cannot be seen or edited after you finish defining the endpoint. To change these values you must delete and recreate the endpoint definition. Expand the endpoint to see the security information.



## Azure service endpoint

Defines and secures a connection to a Microsoft Azure subscription.

Parameter	Description
[authentication type]	Required. Select <b>Credentials</b> , <b>Certificate based</b> , or <b>Service Principal Authentication</b> .
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your Azure account or subscription.
Subscription ID	Required. The GUID-like identifier for your Azure subscription (not the subscription name). You can copy this from the Azure portal.
Subscription Name	Required. The name of your Microsoft Azure subscription (account).
User name	Required for Credentials authentication. User name of a work or school account (for example @fabrikam.com). Microsoft accounts (for example @live or @hotmail) are not supported.
Password	Required for Credentials authentication. Password for the user specified above.
Management Certificate	Required for Certificate based authentication. Copy the value of the management certificate key from your publish settings xml file or the Azure portal.

Service Principal ID	Required for Service Principal authentication. The Azure Active Directory client ID of the account.
Service Principal Key	Required for Service Principal authentication. The Azure Active Directory client key of the account.
Tenant ID	Required for Service Principal authentication. The ID of the client tenant in Azure Active Directory.

## Generic service endpoint

Defines and secures a connection to any other type of service or application.

Parameter	Description
Connection Name	Required. The name you will use to refer to this endpoint in task properties. This is not the name of your account or subscription with the service.
Server URL	Required. The URL of the service.
User name	Required. The username to connect to the service.
Password/Token Key	Required. The password or access token for the specified username.

### 6.2.3.3 Add-in tasks and endpoints

Other types of tasks and the appropriate service endpoint types can be installed in Team Services and Team Foundation Server. Some of the additional task and service endpoint combinations currently available are:

- **Apache Tomcat Deployment.** Deploy your Java web applications to a tomcat server from Visual Studio Team Services or Team Foundation Server.
- **SCVMM Integration:** Connect to a System Center Virtual Machine Manager (SCVMM) server to easily provision virtual machines and perform actions on them such as managing snapshots, starting and stopping VMs, and more.
- **VMware Resource Deployment:** Connect to a VMware vCenter Server from Visual Studio Team Services or Team Foundation Server to provision, start, stop, or snapshot VMware virtual machines.

### 6.2.3.4 Pre-defined variables

The parameter values you use in your tasks can include the following pre-defined variables, which are built-in properties.

System variables (common across Team Services and Team Foundation Server):

Variable name	Description
System.DefaultWorkingDirectory	The working directory of the running job, either the working directory for builds or the working directory for releases.

System.TeamFoundationServerUri	The URL of the Team Foundation Server or Visual Studio Team Services server.
System.TeamFoundationCollectionUri	The URL of the Team Foundation collection or Visual Studio Team Services collection.
System.TeamProject	The name of the team project to which this build or release belongs.
System.TeamProjectId	The ID of the team project to which this build or release belongs.

#### 6.2.3.5 Global release variables

Variable name	Description
Release.DefinitionName	The name of the release definition to which the current release belongs.
Release.EnvironmentUri	The URI of environment to which deployment is currently in progress.
Release.EnvironmentName	The name of environment to which deployment is currently in progress.
Release.ReleaseDescription	The text description provided at the time of the release.
Release.ReleaseId	The identifier of the current release record.
Release.ReleaseName	The name of the current release.
Release.ReleaseUri	The URI of current release.
Release.RequestedFor	The display name of identity that triggered the release.
Release.RequestedForId	The ID of identity that triggered the release.

#### 6.2.3.6 Agent variables

Variable name	Description
Agent.HomeDirectory	The folder where the agent is installed. This folder contains the code and resources for the agent.
Agent.JobName	The name of the job that is running, such as Release or Build.
Agent.MachineName	The name of the computer on which the agent is configured.
Agent.Name	The name of the agent as registered with the agent pool. This is likely to be different from the computer name.
Agent.RootDirectory	The root directory for this agent, used to synchronize with the source.

Agent.WorkingDirectory	The working directory for this agent.
------------------------	---------------------------------------

#### 6.2.3.7 Agent release variables

Variable name	Description
Agent.ReleaseDirectory	The local path configured for the agent, where all folders for a specific release definition are created. It will be \${Agent.RootDirectory}{hash of collection of definition identifier and repository URL}.

## 6.2.4 Create and deploy releases

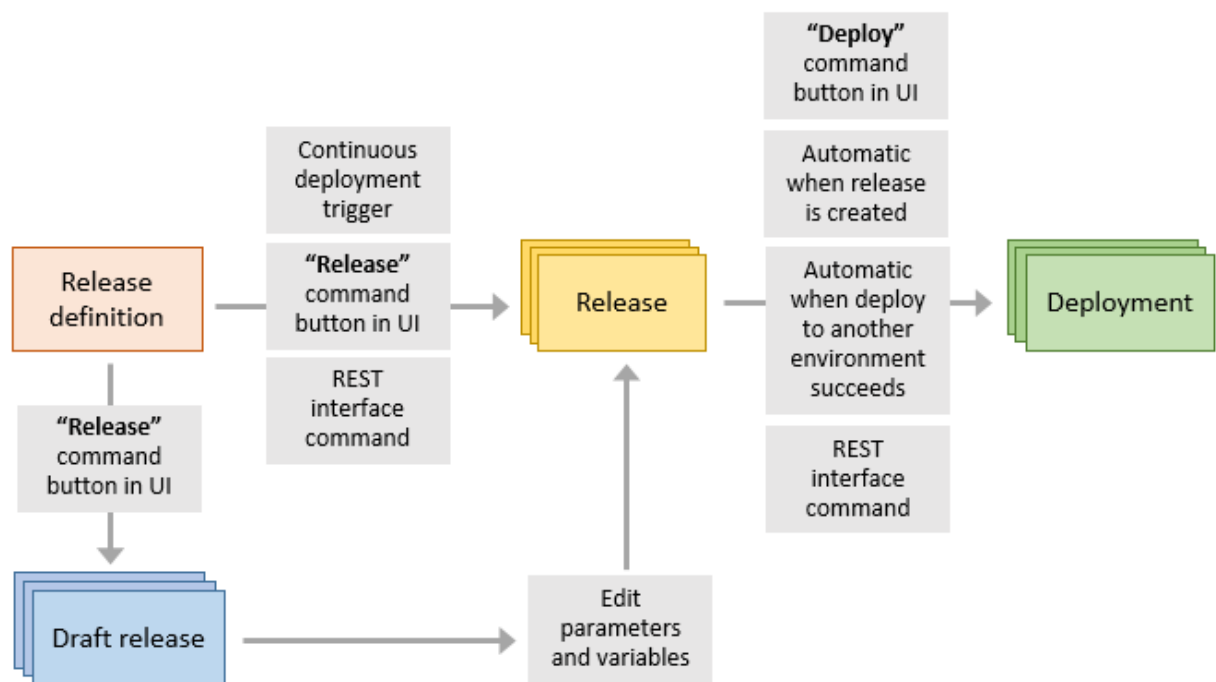
### 6.2.4.1 Understanding releases and deployments

In Release Management, the terms *release* and *deployment* have very different meanings:

- A **release** is the package or container that holds a versioned set of artifacts specified in a **release definition**. It includes a snapshot of all the information required to carry out all the tasks and actions in the release definition, such as the **environments**, the task steps for each one, the values of task parameters and variables, and the release policies such as triggers, approvers, and release queuing options. There can be multiple releases from one released definition, and information about each one is stored and **displayed** in Release Management for the specified **retention period**.
- A **deployment** is the action of running the **tasks** for one environment, which results in the application **artifacts** being deployed, tests being run, and whatever other actions are specified for that environment. A release creates, initializes, and starts each deployment based on the settings and policies defined in the original release definition. There can be multiple deployments from each release.

The following schematic shows the relationship between release definitions, releases, and deployments.





Releases (and, in some cases, draft releases) can be created from a release definition in several ways:

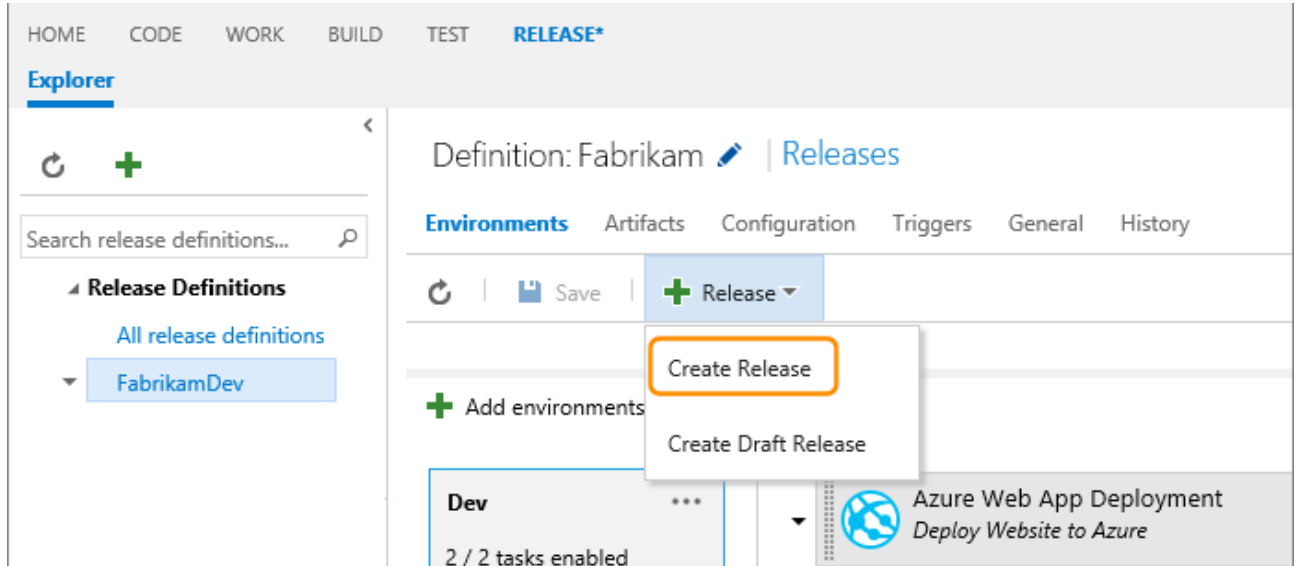
- By a **continuous deployment trigger** that creates a release when a new version of the source build artifacts is available
- By using the **Release** command in the UI to **create a release manually**
- By sending a command over the network to the **REST interface**

**However**, the action of creating a release **does not** mean it will automatically or immediately start a deployment. For example:

- There may be **deployment triggers** defined for an environment, which force the deployment to wait; either for a manual deployment, a network request to the **REST interface**, or for successful deployment to another environment.
- A deployment started manually from the **Deploy command** in the UI, or from a network command sent to the **REST interface**, may **specify a final target environment** other than the last environment in a release pipeline. For example, it may specify that the release is deployed only as far as the QA environment and not to the production environment.
- There may be **queuing policies** defined for an environment, which specify which of multiple deployments will occur, or the order in which releases are deployed.
- There may be **pre-deployment approvers** defined for an environment, and the deployment will not occur until all necessary **approvals have been granted**.
- Approvers may **defer the release** to an environment until a specified date and time.

#### 6.2.4.2 Creating a release from the RELEASE hub

You will need to create a release manually if you have not specified a **continuous deployment or continuous integration trigger** in your release definition. When you are ready to manually create a release from a release definition, open the definition and choose **Create Release** from the **Release** drop-down list.



In the **CREATE NEW RELEASE** dialog, select the version of the linked build artifacts you want to include in this release. Optionally, enter a description for this release.

The screenshot shows the 'CREATE NEW RELEASE FOR FABRIKAMDEV' dialog box. It has a 'Release Description' field with the text 'New manually triggered release'. Below is the 'Artifacts' section with a table. The table has two columns: 'Source name' and 'Version'. The 'Source name' column has the value 'FabrikamDev (Build)'. The 'Version' column has a dropdown menu with '20160127.1' selected and highlighted by an orange box. Below the table is the 'Automated deployments' section with a yellow warning message: 'No deployment will be triggered automatically after release creation. Set the required condition on each environment.' Below that is the 'Manual deployments' section with a label 'Environments on which deployments will have to be manually triggered:' and a list of environments: 'Dev, QA, Prod'. At the bottom right are 'Create' and 'Cancel' buttons.

Notice that the dialog contains two sections named **Automated deployments** and **Manual deployments**. In the case (above) where there are no **environment deployment triggers** set, the **Manual deployments** section shows a list of all the environments. Users will be required to manually initiate deployment to each of these.



*Specifying manual deployment for an environment is one way to prevent a deployment happening until you are sure it is ready to go. However, you can also use approvals at intermediate stages to pause a release and allow it to be cancelled before it reaches the target or final environment.*

However, if you have specified any **environment deployment triggers** in the environments in your definition, the CREATE NEW RELEASE dialog will show this in the **Automated deployments** and **Manual deployments** sections. For example, the following screenshot shows how the dialog appears if you have set the following environment deployment triggers:

- **Dev** environment: After release creation
- **QA** environment: After successful deployment to the **Dev** environment
- **Production** environment: No automated deployment, manual only

CREATE NEW RELEASE FOR FABRIKAMDEV

Release Description: New release of development app

**Artifacts**

Source name	Version
FabrikamDev (Build)	20160127.1

**Automated deployments**

Environments to which deployments will be triggered automatically. You can choose to disable automated deployment to an environment.

Environment	Deployment trigger
Dev	Automated deployment: After release creation
QA	Automated deployment: After successful deployment to Dev

**Manual deployments**

Environments on which deployments will be manually triggered:

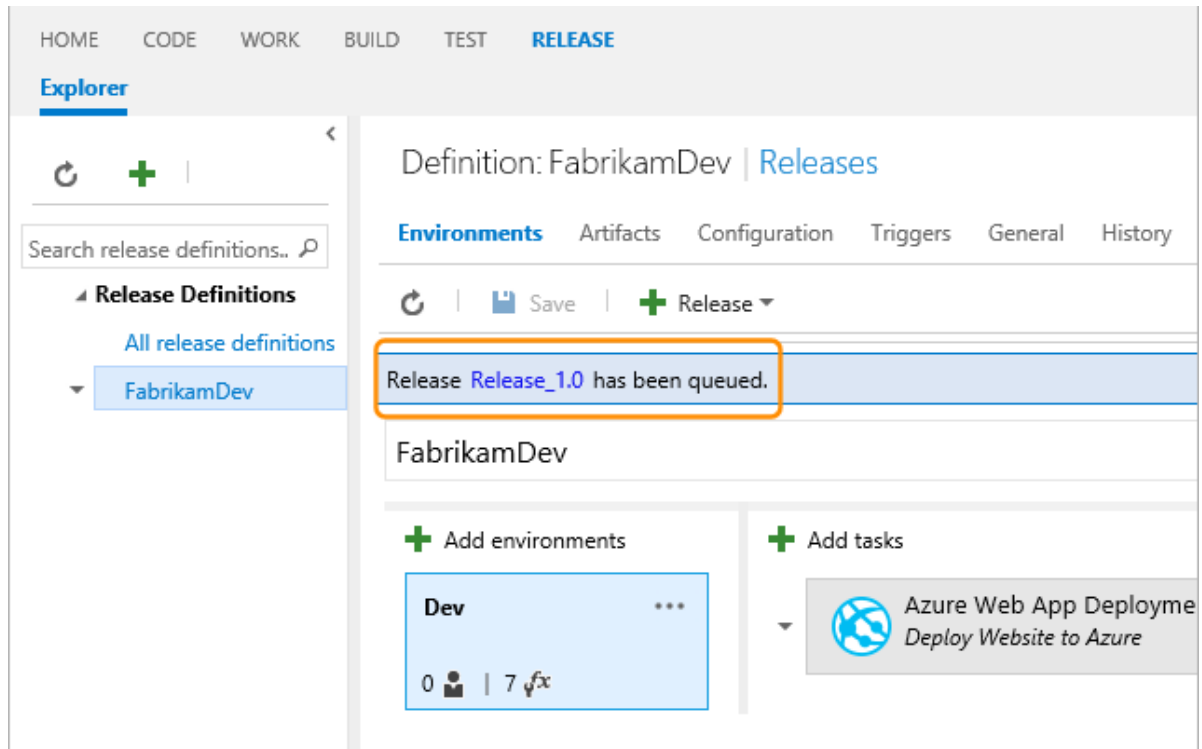
Production
------------

Create Cancel

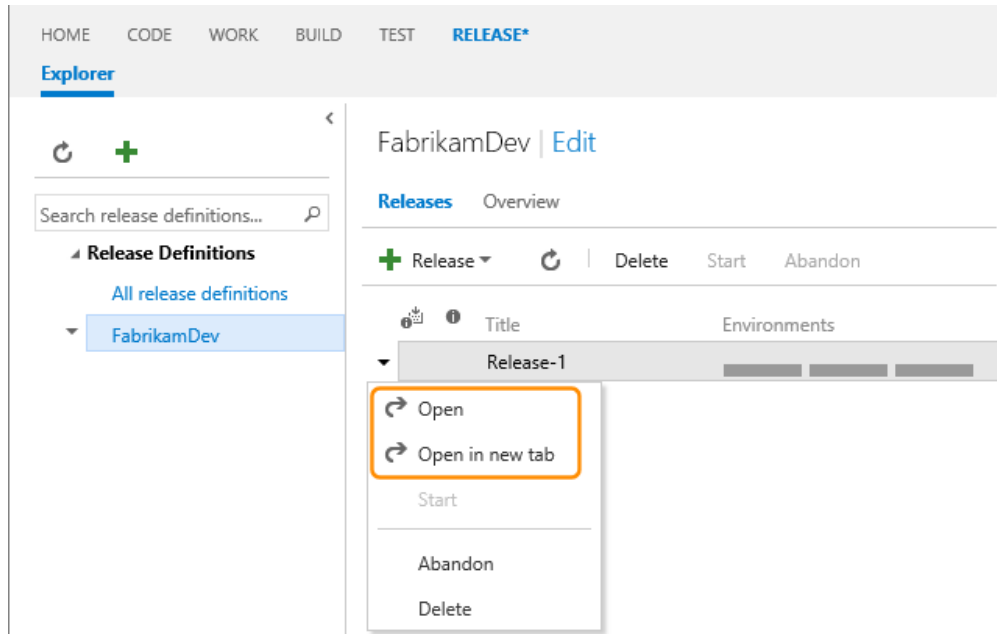
A drop-down list for the environments that have a deployment trigger configured shows the current trigger setting, such as **After release creation** or **After successful deployment to the Dev environment**. Users with permission to edit a release definition can change this setting for just this release.

For example, the setting for the **QA** environment could be changed to **Manual deployment** if this particular release needed to be help back and not automatically deployed to that environment for some reason. This flexibility can reduce the number of release definitions that your team needs to create.

When you are finished editing the values in the dialog, choose **Create**. If there are no errors, the release will be created and you will see this in the message bar. To see details of the release, choose the release name hyperlink.



Alternatively, in **Releases** view, the new release will be shown in the list. Open the shortcut menu for the new release and choose **Open** or **Open in new tab** (or double-click on it) to see details of the release.



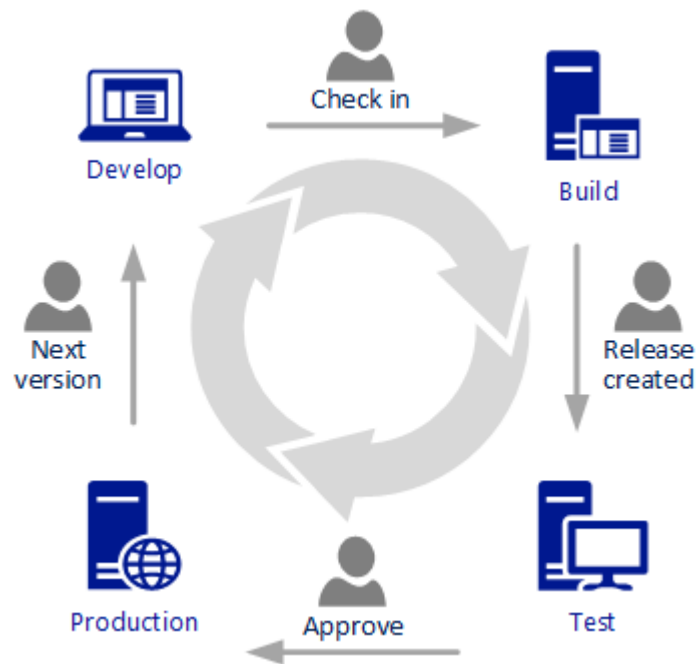
Notice that, in this case, deployment of the release artifacts has not started - all the bars in the list are gray, which means you must now manually start each deployment. If the first environment had a deployment trigger setting of **After release creation**, the first bar would show blue for queued or in-progress (or pending awaiting approval if you have defined a pre-deployment approver for that environment).

The message bar in the release definition page and the menu in the **Releases** page both open the **Summary** view for the newly created release. For details of how you initiate deployments using the **Summary** view, see **Deploying a release**.

#### 6.2.4.3 Creating a release from the BUILD hub

Most build systems support *continuous integration* or *continuous deployment* where simply checking in an updated file can trigger a build. This build might include tasks to deploy the built artifacts to test, QA, and even production servers. However, Release Management offers many useful (if not vital) features for managing and tracking deployments through the use of release definitions and releases.

A release definition in Release Management can be configured to create a new release automatically on completion of a build and, in addition, deploy that release automatically to one or more environments. For example, this allows new builds to be automatically managed, deployed, and tracked all the way from development, through test and QA, to production servers.



However, you may not always want to take advantage of this full integration capability. For example, you may prefer to start builds manually, or create a release manually after a build is complete. Release Management enables you to implement both automated and manual processes to deploy your apps.

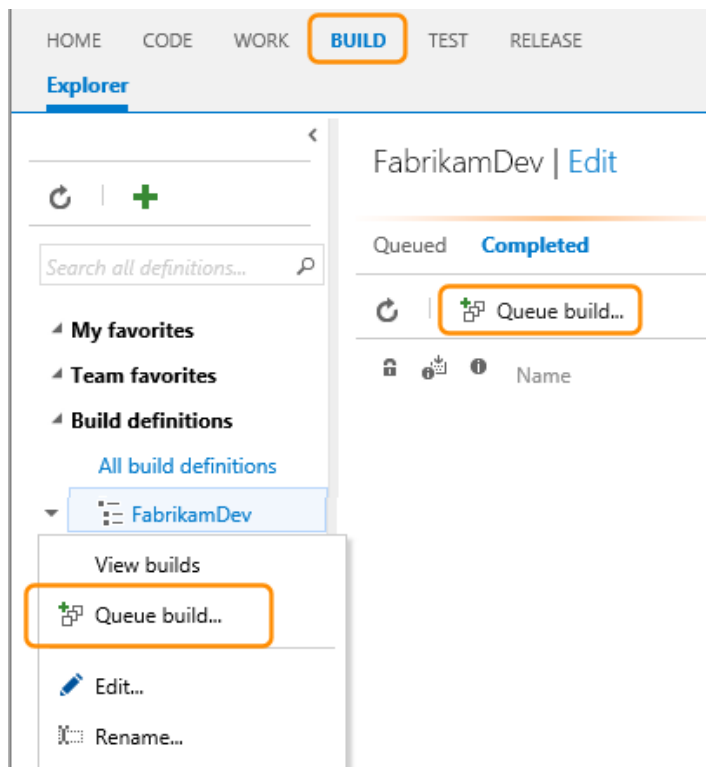
To configure a release definition in Release Management to deploy your application automatically as part of a continuous deployment or continuous integration process, you must set the continuous deployment trigger.

If you **have not** set the **continuous deployment trigger**, or you have not defined a fully orchestrated pipeline for deploying the artifacts specified in your releases, you will need to manually initiate the deployment for some environments. This might be appropriate in some circumstances, such as deploying a final release to a production environment.

#### 6.2.4.4 Starting a continuous integration build to create a release

If you **have** set the **continuous deployment trigger** in your release definition, a new release will be created automatically for any release definition that is linked to that build artifact. The new release(s) will use the new version of the artifact.

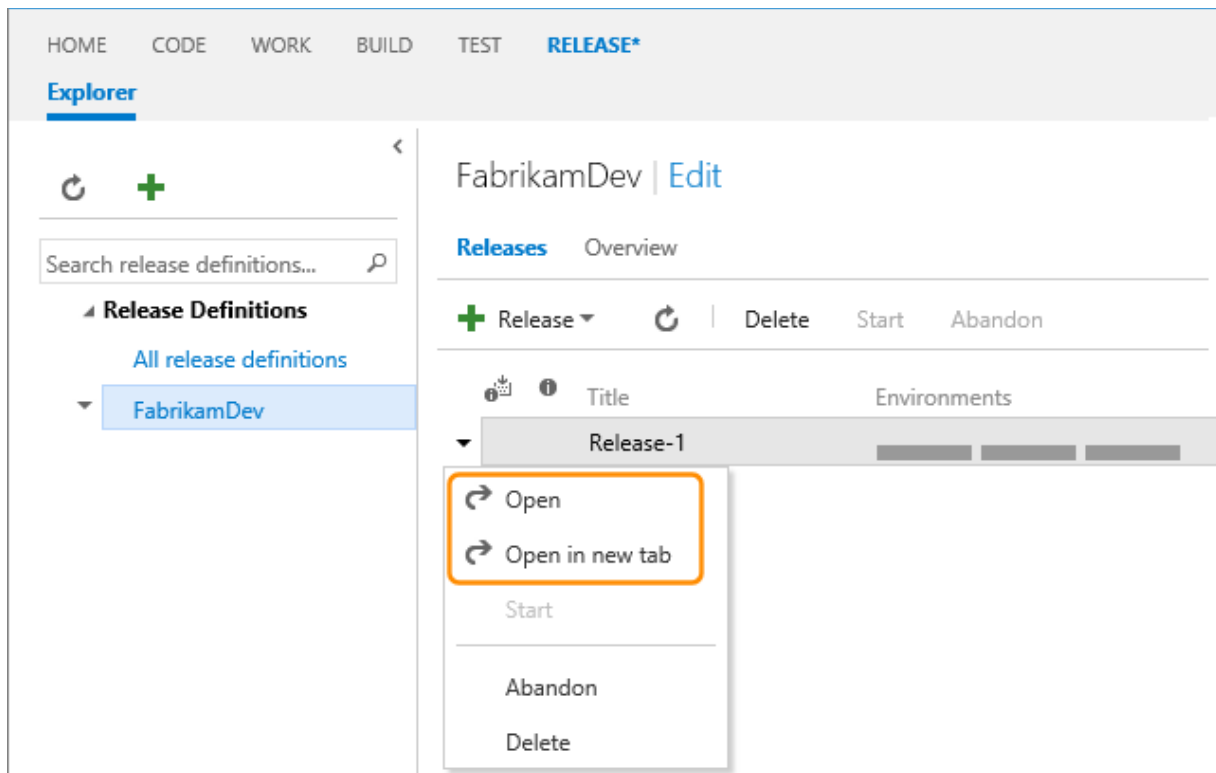
Open the **BUILD** page, select your build definition and choose **Queue build** in the toolbar, or open the shortcut menu for your build definition, and choose **Queue build**.



In the **QUEUE BUILD** dialog, choose **OK** to start the build.

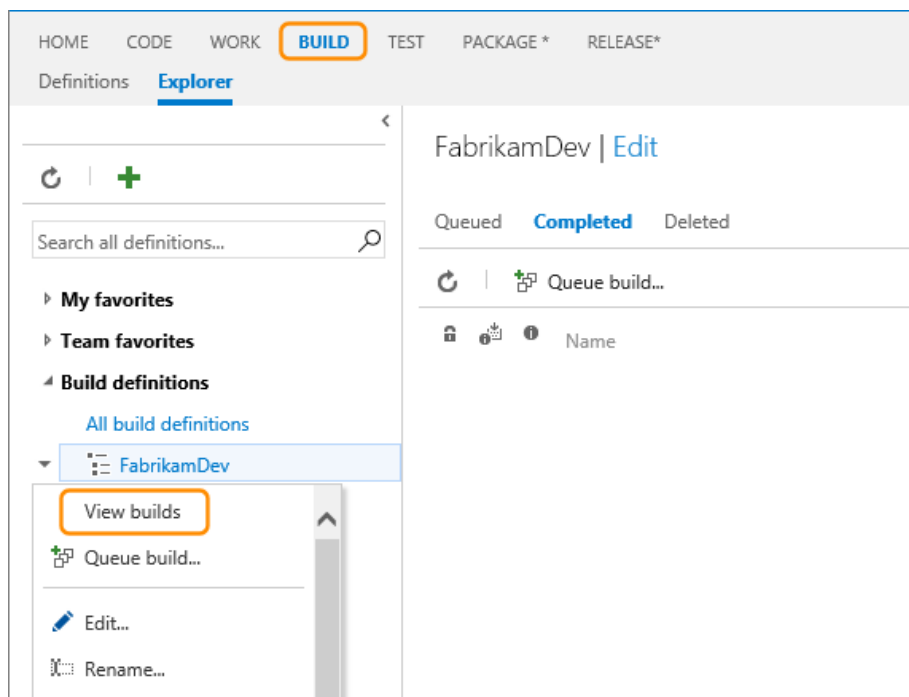
The screenshot shows the 'QUEUE BUILD FOR FABRIKAMDEV' dialog box. It has a title bar with a close button (X). The dialog contains several sections: 'Queue' with a dropdown menu set to 'Hosted', 'Branch' with a dropdown menu set to 'master', and 'Commit' with a text input field. Below these is a section for 'Variables' and 'Demands'. Under 'Variables', there are two rows: 'BuildConfiguration' with a text input set to 'debug', and 'BuildPlatform' with a text input set to 'any cpu'. Below these is a green plus sign icon followed by the text 'Add variable'. At the bottom right, there are two buttons: 'OK' (highlighted) and 'Close'.

If the build succeeds, the release will be created in exactly the same way as if you started it manually in the **RELEASE** hub. To view the status of the release, or to initiate deployment to specific environments, open the **RELEASE** hub. In the **Releases** view, open the shortcut menu for the new release and choose **Open** or **Open in new tab** (or double-click on it) to open the **Summary** view.



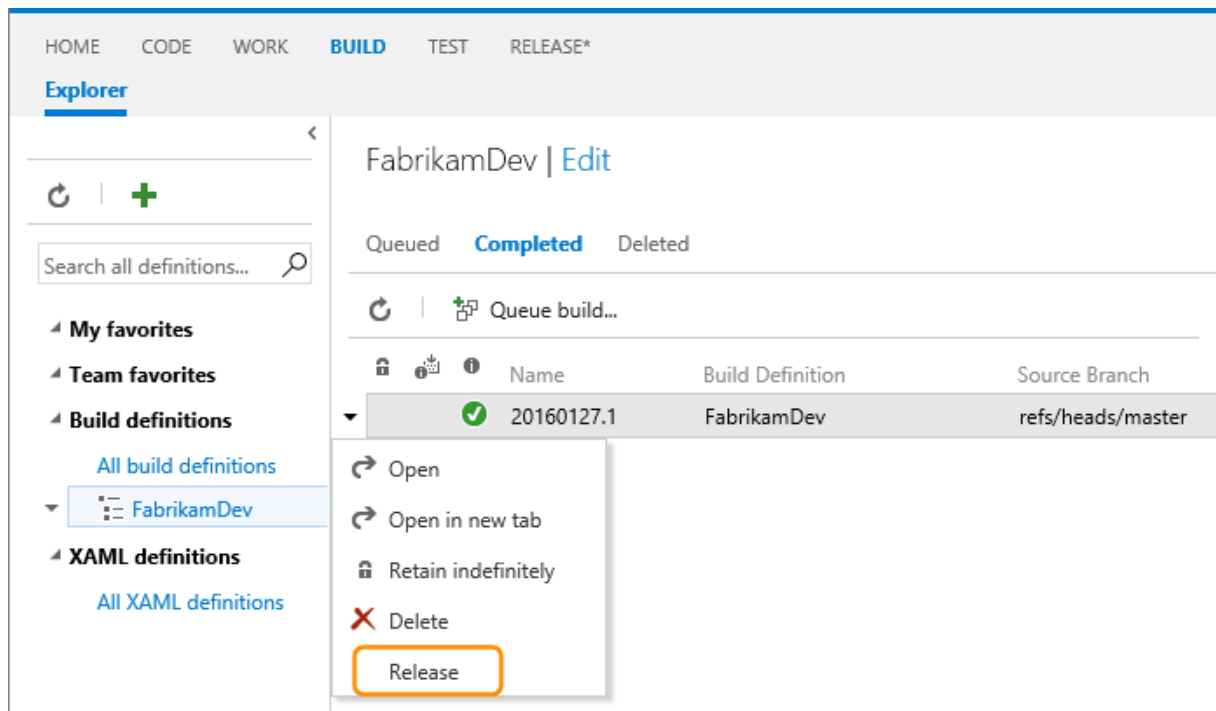
#### 6.2.4.5 Creating a release from a build result

You can create a release, or view the status of a release, from a build summary view. In the **BUILD** hub, open the shortcut menu for a build definition and choose **View builds**.





In the **Explorer** tab list of build results, select a build, open the shortcut menu, and choose **Release**.



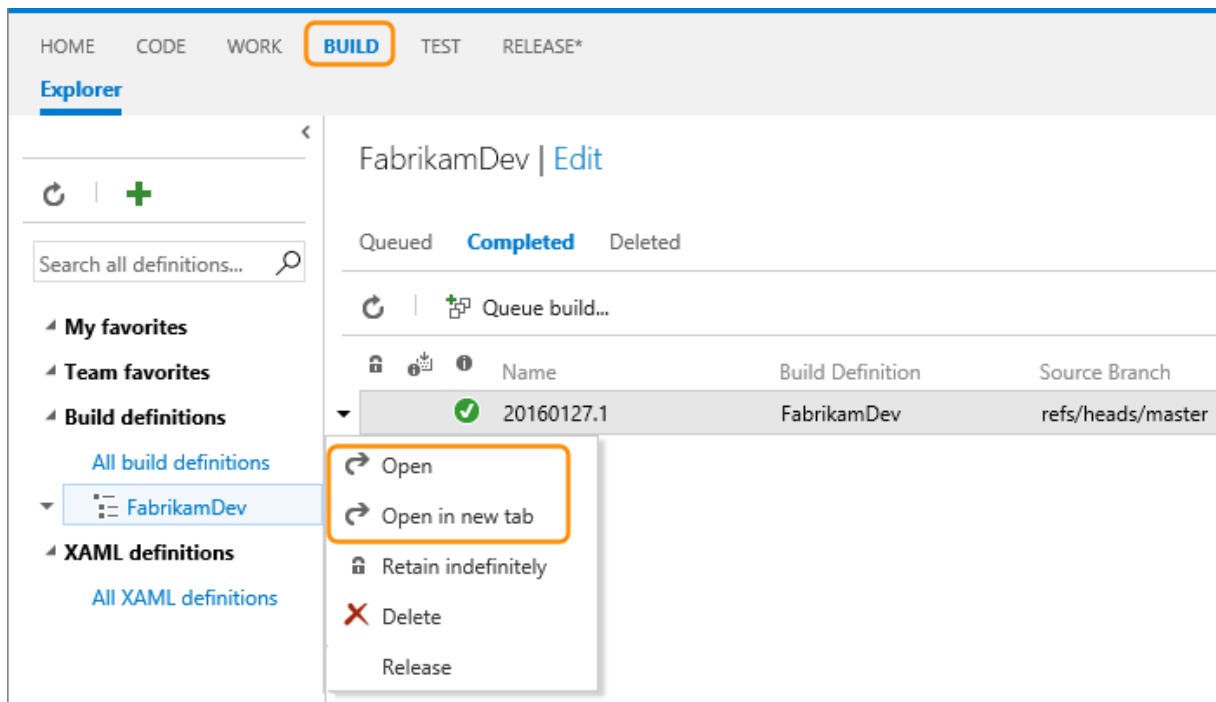
The **CREATE NEW RELEASE** dialog shows a list of release definitions that can be used (the release definitions linked to this build definition), the version of the build artifacts, and the environments to which the artifacts will be deployed. Choose **Create** to create the new release.

The 'CREATE NEW RELEASE' dialog box is shown. It has a title bar with a close button. The dialog is divided into several sections:

- Release definition:** A dropdown menu showing 'FabrikamDev'.
- Release Description:** A text box containing 'Release initiated from Build hub'.
- Artifacts:** A table with two columns: 'Source name' and 'Version'. The first row shows 'ASPNet4.CI (Build)' and '20160127.1'.
- Automated deployments:** A section titled 'Environments on which deployments will be triggered automatically:' followed by a list of environments: 'Dev, QA'.
- Manual deployments:** A section titled 'Environments on which deployments will have to be manually triggered:' followed by a list of environments: 'Prod'.

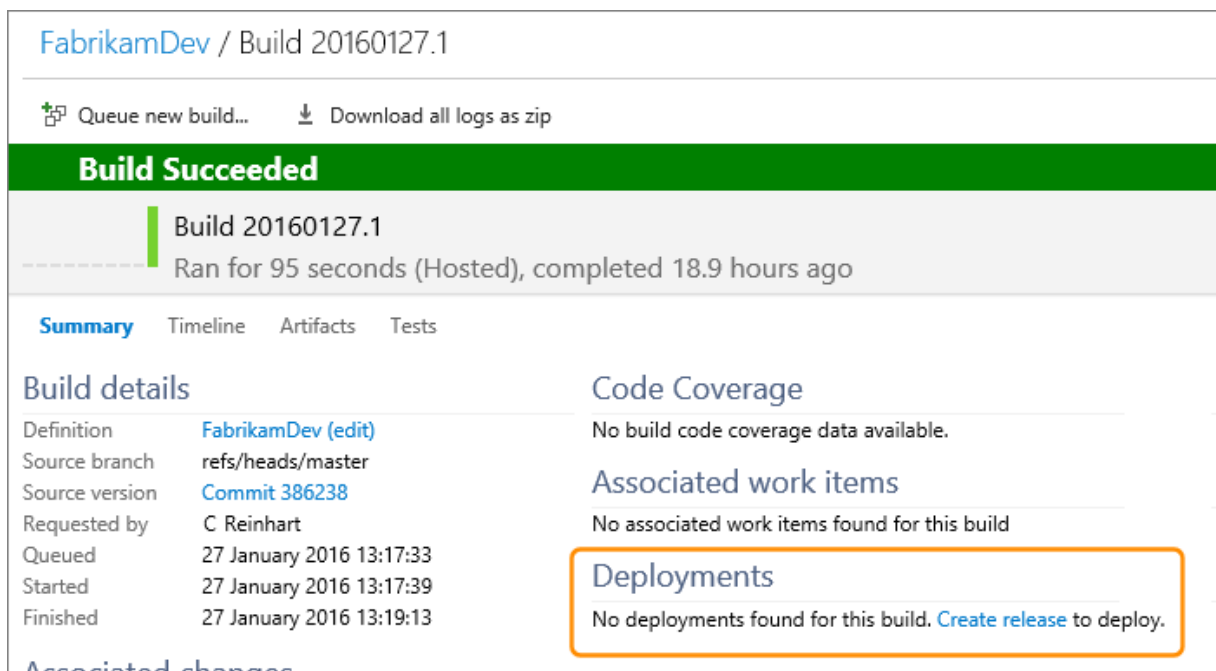
At the bottom right, there are two buttons: 'Create' and 'Cancel'. The 'Create' button is highlighted with a blue border.

Alternatively, select a build in the list, open the shortcut menu, and choose **Open** or **Open in new tab** (or double-click on the build).



The screenshot shows the 'BUILD' tab in the Visual Studio interface. On the left, the 'Explorer' pane shows a tree view with 'FabrikamDev' selected under 'Build definitions'. A context menu is open over the build '20160127.1' in the main pane. The menu options are: 'Open', 'Open in new tab', 'Retain indefinitely', 'Delete', and 'Release'. The 'Open' and 'Open in new tab' options are highlighted with an orange box.

If the build has not yet been deployed, the **Deployments** section contains a link **Create release**.



The screenshot shows the 'FabrikamDev / Build 20160127.1' page. The page has a green banner that says 'Build Succeeded'. Below the banner, there is a 'Summary' tab selected, showing build details. The 'Deployments' section is highlighted with an orange box, showing 'No deployments found for this build. Create release to deploy.'

Choose the **Create release** link to create a new release from this build. The CREATE NEW RELEASE dialog shows a list of release definitions that can be used (the release definitions linked to this build definition), the version of the build artifacts, and the environments to which the artifacts will be deployed. Choose **Create** to create the new release.

CREATE NEW RELEASE

Release definition

FabrikamDev

Release Description

Release initiated from Build hub

Artifacts

Source name	Version
ASPNet4.CI (Build)	20160127.1

Automated deployments

Environments on which deployments will be triggered automatically:

Dev, QA

Manual deployments

Environments on which deployments will have to be manually triggered:

Prod

Create

Cancel

If the selected build has previously been deployed, the build status view shows the environments to which it was deployed.

FabrikamDev / Build 20160127.1

Queue new build...

Download all logs as zip

Build Succeeded

Build 20160127.1

Ran for 95 seconds (Hosted), completed 18.9 hours ago

Summary

Timeline

Artifacts

Tests

Build details

Definition	<a href="#">FabrikamDev (edit)</a>
Source branch	refs/heads/master
Source version	<a href="#">Commit 386238</a>
Requested by	C Reinhart
Queued	27 January 2016 13:17:33
Started	27 January 2016 13:17:39
Finished	27 January 2016 13:19:13

Code Coverage

No build code coverage data available.

Associated work items

No associated work items found for this build

Deployments

<div>Dev</div> <div>deployed with Release-4</div>	Succeeded
---	-----------

Choose the link to an environment to open the Release Management **Overview** for this release definition.

HOME CODE WORK BUILD TEST **RELEASE\***

**Explorer**

Search release definitions...

**Release Definitions**

All release definitio

FabrikamDev

FabrikamDev | Edit

Releases **Overview**

+ Release

FabrikamDev ...

Dev QA Prod

Release-4 17 hours ago

Release-4 17 hours ago

No deployments yet

#### 6.2.4.6 Deploying a release

When you create a release from a release definition, you can see the current status of that release in the **Summary** view. In some cases, deployment of the release to all environments may occur automatically through **environment deployment triggers**. However, where this is not the case, you must initiate deployment to environments manually.

The **Environments** section of the **Summary** view lists the environments in the release definition, and shows the deployment state for each one. In this example, they are all gray (not deployed), which means you must manually start each deployment.

FabrikamDev / Release-1

**Summary** Environments Artifacts Configuration General Commits Work items Logs

Save Abandon + Deploy

**Details**

New manually triggered release

Manually created by C Reinhart 3 minutes ago

ASPNet4.CI / 20160127.1 (Build) refs/heads/master

**Work items**

No associated work items found.

**Test results**

No test results available for this release.

**Environments**

Environment	Actions	Deployment Status	Started	Duration
Dev	...	NOT DEPLOY...		
QA	...	NOT DEPLOY...		
Prod	...	NOT DEPLOY...		

**Issues**

No issues reported in this release.

If the first environment has a deployment trigger setting of **After release creation**, the first bar will be blue to show it is queued, in-progress, or pending awaiting approval if you have defined a pre-deployment approver for that environment.

Environments				
Environment	Actions	Deployment Status	Started	Duration
Dev	...	PENDING		
QA	...	NOT DEPLOY...		
Prod	...	NOT DEPLOY...		

You start the deployments to any environments that have not been deployed by opening the shortcut menu from the ellipses (...) in the **Actions** column and choosing **Deploy**.

Environments				
Environment	Actions	Deployment Status	Started	Duration
Dev	...	NOT DEPLOY...		
QA				
Prod				
Issues				

...

Deploy

Cancel

Redeploy

Alternatively, choose the environment you want to deploy to from the **Deploy** drop-down list in the toolbar.

FabrikamDev / Release-1

Summary
Environments
Artifacts
Configuration
General
Commits
Work items
Logs

Refresh
Save
Abandon
+ Deploy

Details

Release-3  
Manually created by C Reinhart 20...  
ASPNet4.CI / 20160127.1 (Build)

Work items

No associated work items found.

Test results

No test results available for this release.

Environments

Environment	Actions	Deployment Status	Started	Duration
Dev	...	SUCCEEDED	19 minutes ago	00:03:30
QA	...	PENDING	16 minutes ago	
Prod	...	NOT DEPLOY...		

Notice that the list shows only the environments that are available for deployment. In this example the **Dev** environment deployment already succeeded, the **QA** deployment is in progress, and so the **Deploy** list shows only the **Prod** environment.

Both the drop-down list and the ellipses shortcut menu commands display the **Deploy** dialog, which contains extensive information about the deployment you are about to start.

### Deploy Release-2 on Dev

1

0

2

2

#### Artifacts of Release-2

ASPNet4.CI / 20160127.1 (Build) refs/heads/master

#### Work items

Comparing Release-2 with Release-1 (currently on Dev)

No associated work items found.

#### Commits

Comparing Release-2 with Release-1 (currently on Dev)

16 June 2015

- CR Updated FabrikamFiber.CallCenter.sln  
386238 by Chuck Reinhart, 16/06/2015

#### Approvers

Pre-deployment approver(s)  
AH Chuck Reinhart

Post-deployment approver(s)  
Chuck Reinhart

#### Tasks

Deploy

Cancel

The five sections of this dialog, which you can scroll to using the icons on the left are:



**Artifacts.** Details of the artifacts that are included in this release for deployment to this environment.



**Work items.** A list of work items (if any) associated with changes to the code for this release.



**Commits.** A list of repository commits (if any) associated with changes to the code for this release.



**Approvers.** A list of the approvers for this environment, as defined in the release definition.



**Tasks.** Details of the task steps that will be executed for the deployment to this environment.

When you are happy that everything is correct, choose **Deploy** to start the deployment to this environment.

#### 6.2.4.7 Pausing or terminating a release

Even though a release can contain more than one environment in its pipeline, it doesn't necessarily mean that the application will be automatically deployed to all of the environments in the release definition. If you configure approvers for the environments in the release pipeline, the release will pause and wait for approval at these stages. Approval can be granted or denied.

HOME CODE WORK BUILD TEST **RELEASE\***

**Explorer**

Search release definitions...

**Release Definitions**

All release definitions

**FabrikamDev**

**FabrikamDev | Edit**

**Releases** Overview

+ Release | Delete | Start | Abandon

Title	Environments	Build
Release-1		20160216.1

In addition, when you create a release from a release definition (in other words, when you initiate a release) where the environments do not have deployment triggers defined, you must start the deployment manually and at that point you can choose which environments to deploy to.

**FabrikamDev / Release-1**

**Summary** Environments Artifacts Configuration General Commits Work items Logs

Refresh | Save | Abandon | + Deploy

**Details**

New manually triggered release

Manually created by C Reinhart 3 minutes ago

ASPNet4.CI / 20160127.1 (Build) refs/heads/master

**Work items**

No associated work items found.

**Test results**

No test results available for this release.

**Environments**

Environment	Actions	Deployment Status	Started	Duration
Dev	...	NOT DEPLOY...		
QA	...	NOT DEPLOY...		
Prod	...	NOT DEPLOY...		

**Issues**

No issues reported in this release.

This allows you to test the initial steps and tasks in a release, safe in the knowledge that it will stop before, for example, deployment to a live production environment.

Finally, you can use the commands in the **Summary** view to cancel a pending release.



## 6.2.5 Track deployments

Release Management provides comprehensive traceability and visibility features to help you monitor the progress of releases, view the history, audit processes using log files, and manage the execution of the tasks in a release pipeline through approvals.

### 6.2.5.1 Understanding the overview and list of releases

The default **Releases** page displayed in Release Management shows a list of releases. As well as the current status, title, release definition name, and other details, the list contains information about the source of the release, when the release was started the user who initiated it, the description, and more (not all columns are visible in this screenshot).

Select a release definition in the left column to filter the list to just that release definition. Sort the list in ascending or descending order based on the values in each column by clicking the column headings (the ▼ icon shows the current sorting column). Use the links in the rows to open the release definition in edit mode, or to see the build results linked to that release.

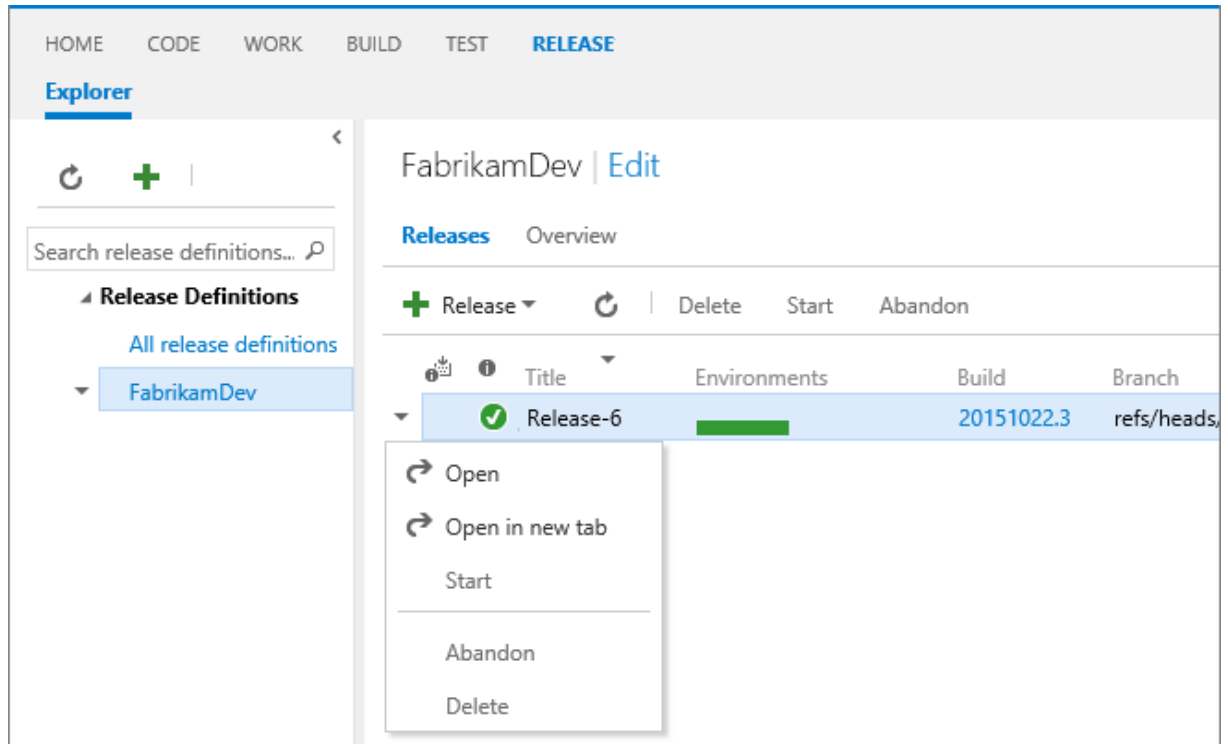
The screenshot shows the 'RELEASE' tab in the top navigation bar. Below it, the 'Explorer' sidebar on the left has 'Release Definitions' expanded, with 'All release definitions' selected. The main content area is titled 'All release definitions' and has tabs for 'Releases' (active) and 'Overview'. Below the tabs is a toolbar with icons for refresh, delete, start, and abandon. The main table lists releases with columns: Title, Release Definition, Environments, Build, and Branch. The first release, 'Release-7', is highlighted and has a play icon in the first column. The other releases are 'Release-6', 'Release-5', and 'Release-4'.

	Title	Release Definition	Environments	Build	Branch
▶	Release-7	FabrikamDev		20151023.5	refs/heads/master
▶	Release-6	FabrikamDev		20151023.1	refs/heads/master
▶	Release-5	FabrikamDev		20151022.3	refs/heads/master
▶	Release-4	FabrikamDev		20151022.2	refs/heads/master

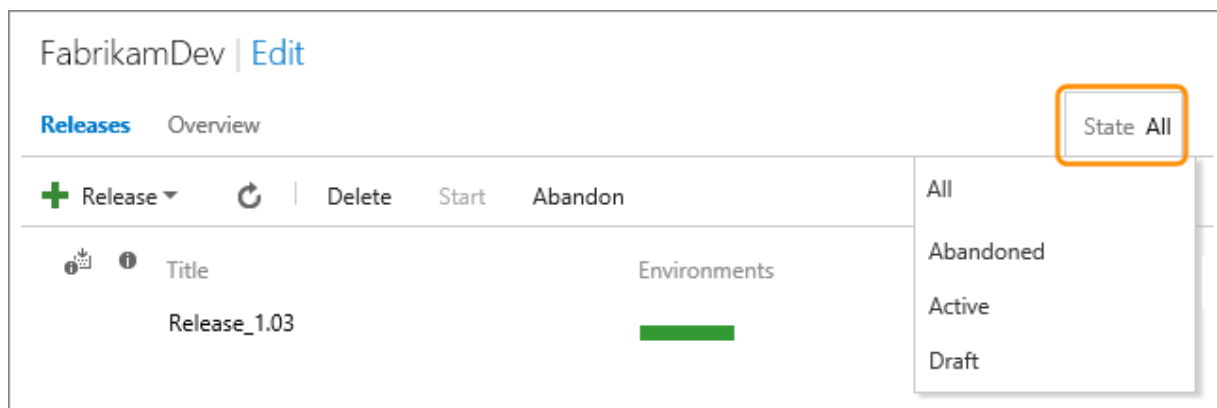
The first column in the list shows the *reason* for each release. The icon indicates that this release was automatically initiated when a new build of the source artifacts was completed - based on the **continuous deployment** setting in the release definition.

In this view, each environment is shown as a horizontal bar - red for failed, green for succeeded, blue for queued or paused, and gray for not deployed. Notice the icon in the first release in the list. This indicates that you were specified as an approver and the release is paused waiting for you to approve it. When a release is paused waiting for another user to grant approval, the list displays the icon.

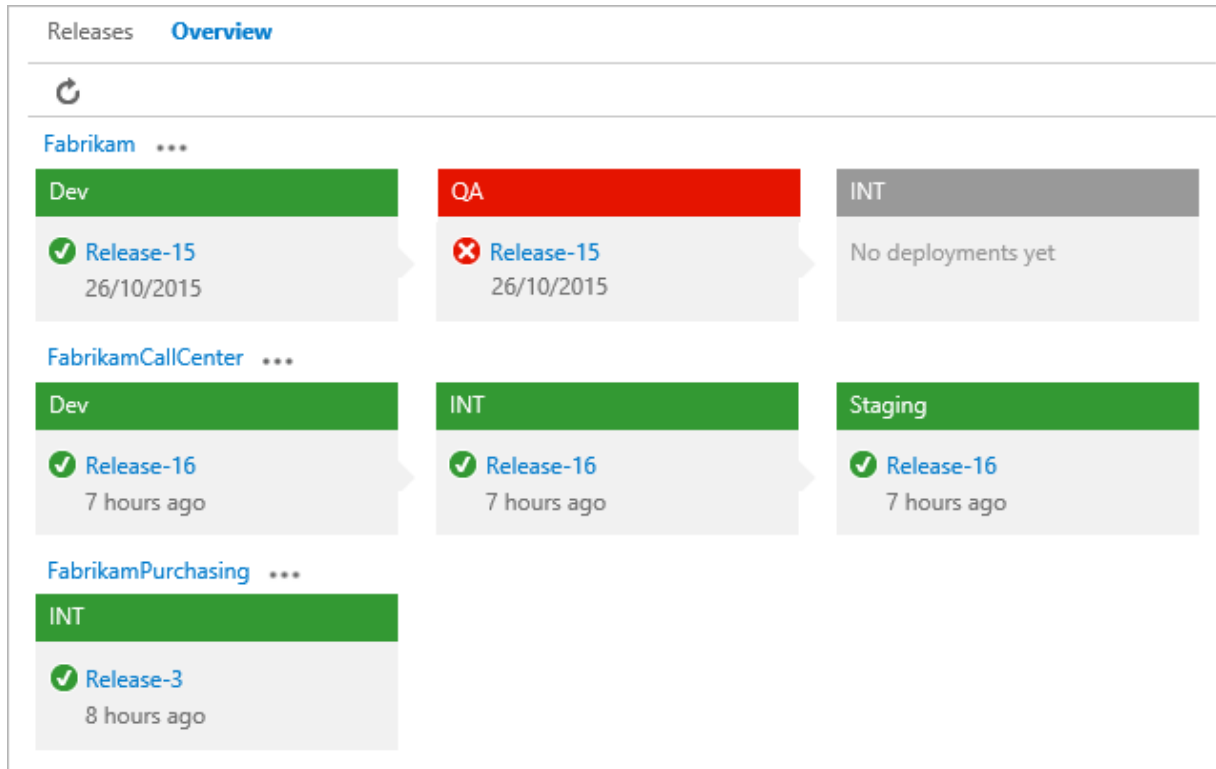
Use the commands on the toolbar of the **Releases** pane, or on the shortcut menu, to **Start** a cancelled or a failed release, **Delete** the details of a release, or **Abandon** a cancelled or a failed release. Depending on the current state of a release, not all of the commands will be available.



Use the drop-down list at the top right of the **Releases** pane to filter the list of releases by their current state.

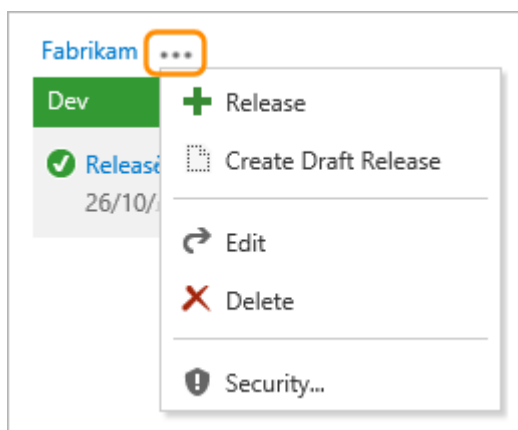


The **Overview** page shows a list of release definitions. Each one is shown as a series of environments, with the name of the release and the date or time it was started. The color of the heading and the icon in each environment indicate the current status of the release. The color scheme is the same as in the **Releases** page. Select a release definition in the left column to filter the list to just releases for that definition.



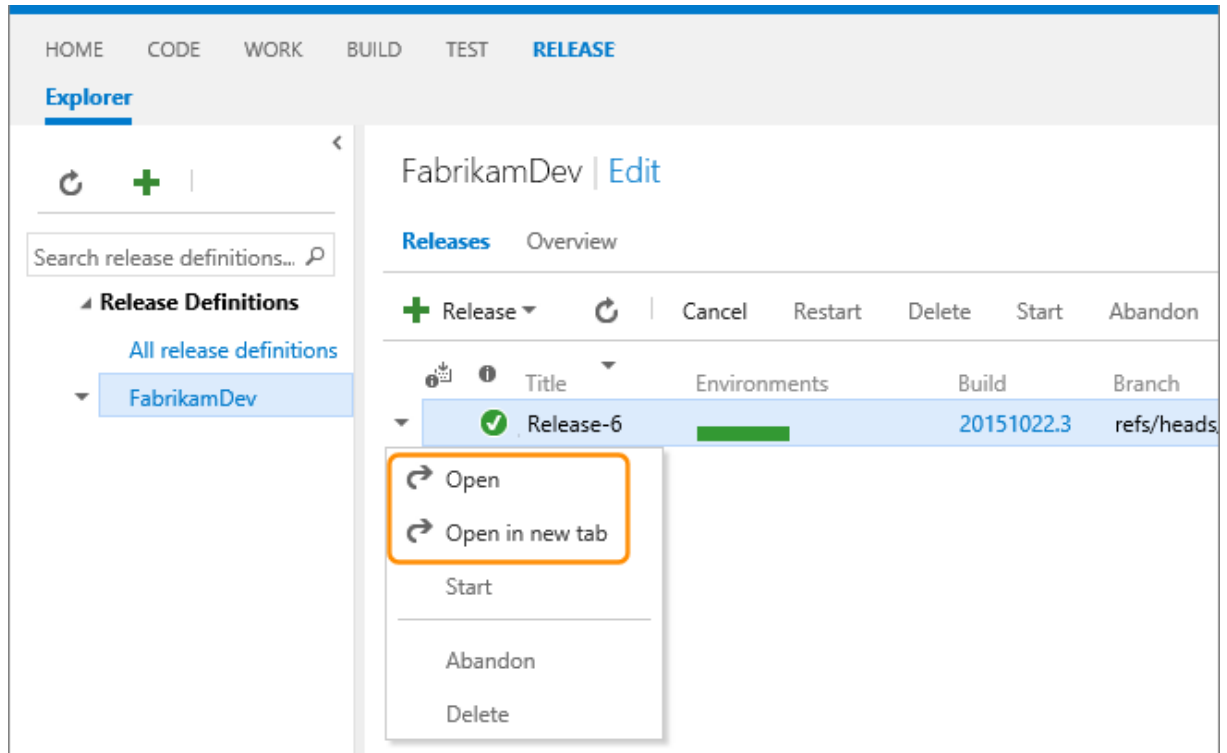
In the **Overview** page, you can:

- Choose the name of the release definition (above the environments) to open that definition in edit mode at the **Environments** page.
- Choose the name of the release displayed within any of the environments to open the **Summary** page for the *most recent* release for this definition.
- Choose the ellipses (...) next to the release definition name to open a shortcut menu where you can create a new release or a draft release for that release definition, edit or delete the release definition, or manage the security settings for the release definition.



#### 6.2.5.2 Understanding the summary view of a release

As a release progresses, you can view the status. You can also view comprehensive details of all previous releases for all release definitions. In the **Releases** list, open the shortcut menu for a release and choose **Open** or **Open in new tab**.



If you have a release in progress, or recently started, a message bar at the top of the page contains a link that opens the **Summary** page for that release.



The default **Summary** tab shows details of the release as it is in progress. The indicator next to the release name shows the current status. The main body of the page shows all of the environments for the release, and the current progress and status for each one. This page also contains information about the release definition and the release itself. This includes details of the artifacts to deploy, any test results, and the work items related to this release (if any).

FabrikamDev / Release-1

Summary Environments Artifacts Configuration General Commits Work Items Logs

| Save | Abandon | Deploy ▼

**Details**

Triggered by FabrikamDev 20151022.2.

Continuous integration requested for C Reinhart 2 minutes ago

FabrikamDev / 20151022.2 (Build) refs/heads/master

**Work Items**

No associated work items found.

**Test Results**

No test results available for this release.

**Environments**

Environment	Actions	Deployment Status	Started	Duration
Dev	...	PENDING	2 minutes ago	
Test	...	NOT DEPLOY...		

**Issues**

No issues reported in this release.

To cancel a pending or in-progress release, use the ellipses (...) for the environment to open the shortcut menu and choose **Cancel**.

FabrikamDev / Release-1

Summary Environments Artifacts Configuration General Commits Work items Logs

| Save | Abandon | Deploy ▼

**Details**

Release-5

Manually created by C Reinhart 3 hours ago

FabrikamDev / 20160127.1 (Build) refs/heads/master

**Work items**

No associated work items found.

**Test results**

No test results available for this release.

**Environments**

Environment	Actions	Deployment Status	Started	Duration
Dev	...	PENDING	3 hours ago	
QA	...		3 hours ago	01:52:22
Production	...		3 hours ago	

**Issues**

To deploy a release that has not yet been deployed, such as releases where the environment deployment options only permit a manual release, use the ellipses (...) for the environment to open the shortcut menu and choose **Deploy**.

FabrikamDev / Release-6

**Summary** Environments Artifacts Configuration General Commits Work items Logs

🔄 | 📁 Save Abandon | + Deploy ▾

---

**Details**

Release-6

Manually created by C Reinhart 55 minutes ago

🏠 FabrikamDev / 20160127.1 (Build) 📁 refs/heads/master

**Work items**

No associated work items found.

---

**Test results**

No test results available for this release.

---

**Environments**

Environment	Actions	Deployment Status	Started	Duration
Dev	⋮	NOT DEPLOY...		
QA	⬆️ Deploy			
Production	❌ Cancel		55 minutes ago	00:00:28
	⬆️ Redeploy			

**Issues**

Choose the **Logs** tab at the top of the page. The **Logs** page shows the current status for each task for each environment, as well as a live view of the logs as the release is progressing.

FabrikamLight / Release-1

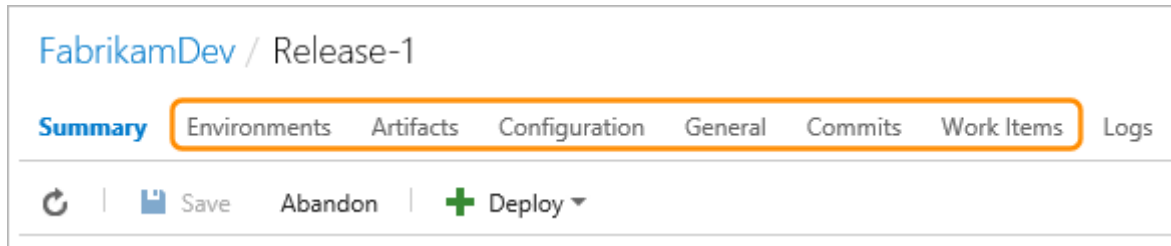
Summary Environments Artifacts Configuration General Commits Work items **Logs**

🔄 | 📁 Save Abandon | + Deploy ▾ | 📄 Download all logs as zip

---

Step	Details	Agent: Hosted Agent Start Time: 12/02/2016 15:31   Durat:
📁 Default Environment		
✅ Pre-deployment approval	👤	
⬆️ Deploy	📄	<pre> ***** Initializing Hosted Agent ***** Requesting an Agent from the Hosted Agent Pool Hosted Agent successfully assigned. Configuring connection settings for Hosted Agent Starting Hosted Agent Initializing connection to Hosted Agent Successfully connected to Hosted Agent ***** Starting: Release ***** Executing the following commandline: C:\LR\MMS\Services\Mms\TaskAgentProvisioner\Tools\agents\d /name:Worker-f18f24da-4fba-49c7-8c57-c832bf5ec3d2 /id:f1f 2 /rootFolder:"C:\LR\MMS\Services\Mms\TaskAgentProvisioner rwarding 1 0 0-Verboesity=Verbose Name=Agent1-1aadeehh9c439d </pre>
✅ Initialize	📄	
✅ Download artifacts	📄	
🔵 Deploy Website to Azure	📄	
🕒 Run Tests	📄	

Use the other tab links to see more information about an in-progress or a completed release.



- Use the **Environments** tab to see details of the release definition used to create this release. The view is read-only, but is useful to examine the configuration and parameters used for this release.
- Use the **Artifacts** tab to see a list of the artifacts deployed as part of this release. You can also open the **Artifacts** page using the **Show all artifacts** link in the **Summary** page.
- Use the **Configuration** tab to see a list of the configuration variables used for this release.
- Use the **General** tab to see the option settings for this release definition, such as the format for release names and the retention period for release information.
- Use the **Commits** tab to see a list of all the source code repository commits associated with this release.
- Use the **Work Items** tab to see a list of all the source code repository work items associated with this release.

#### 6.2.5.3 Approving a release

During a release, the deployment pipeline will stop at any stage that requires approval, and will display an indicator icon in the **Releases**, **Summary**, and **Logs** pages and lists. Release Management may also send an email message to the approver(s) defined for this approval step (this is configurable)

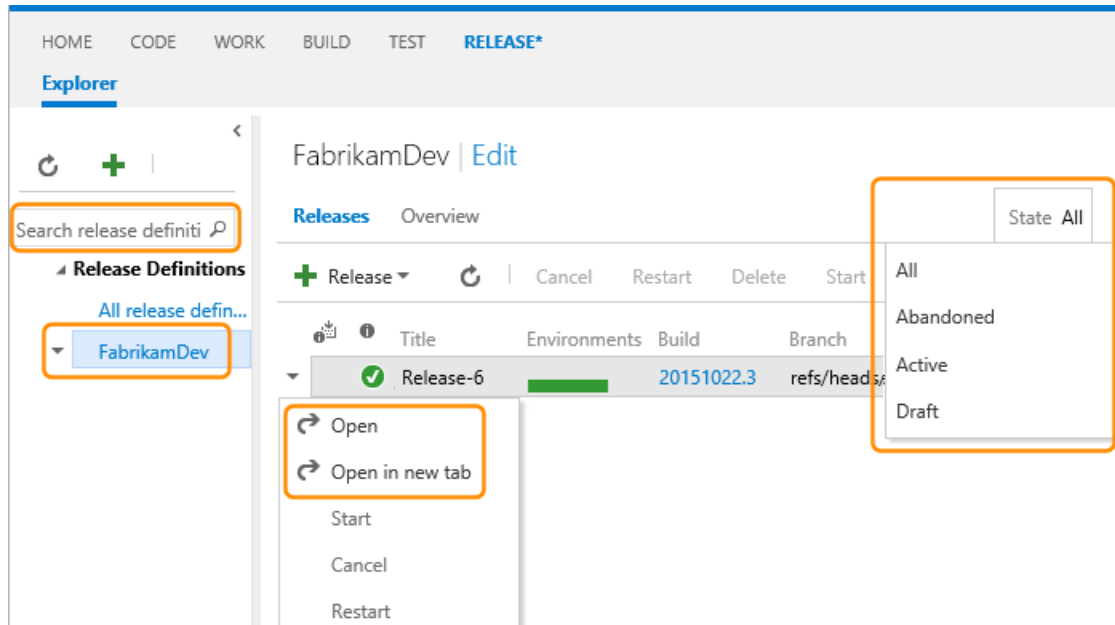
#### 6.2.5.4 Viewing release logs

When you open Release Management, the left column displays a list of all release definitions and the main body of the page shows a list of all releases for all release definitions. To view the logs for a release, start by opening the **Summary** page for the release you're interested in.

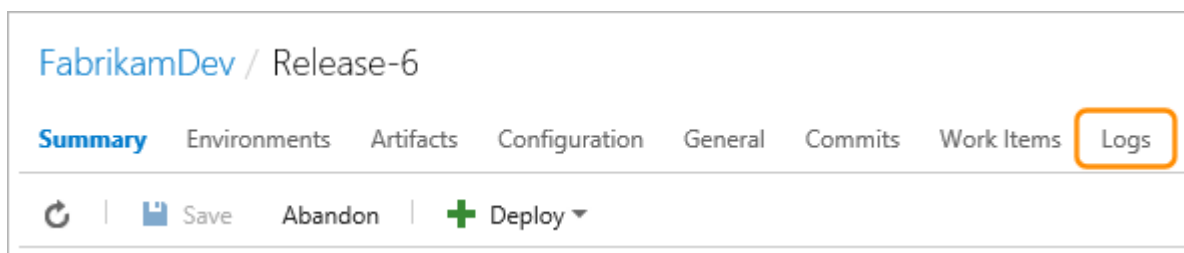
Filter the list of releases by:

- Selecting a release definition in the left column.
- Typing part of a release definition name in the **Search release definitions** textbox.
- Selecting a release state from the drop-down list at the top right of the page.

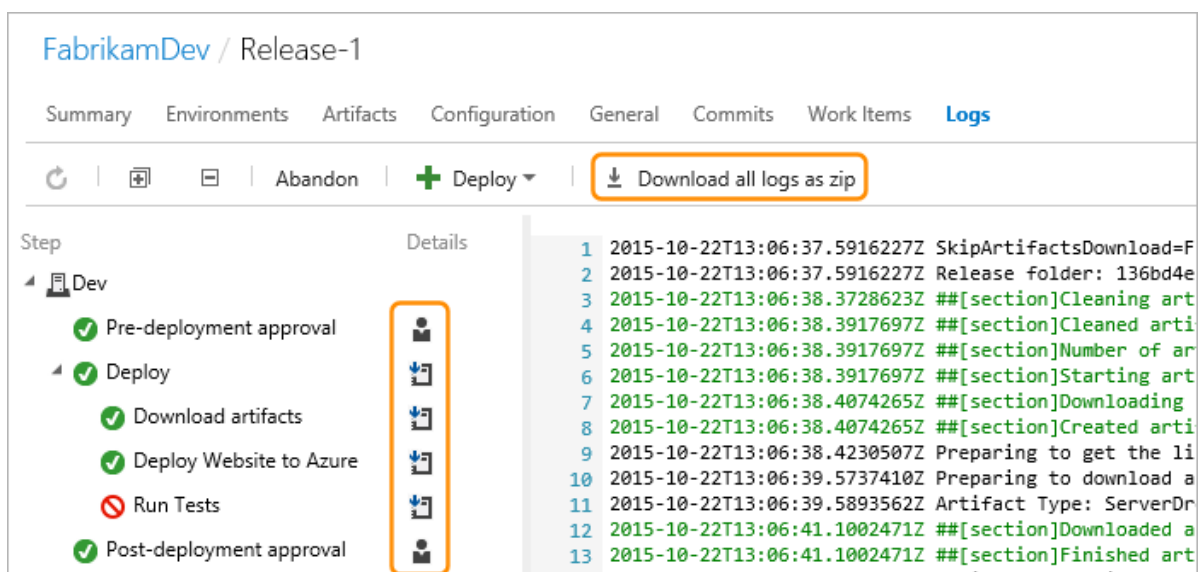
Open the **Logs** page by choosing **Open** or **Open in new tab** from the shortcut menu for a release (or by double-clicking the release).



In the **Summary** page, choose the **Logs** tab link.



The **Logs** page shows the status for each step or task of the release, for each of the environments in the release definition. After a release has completed, irrespective of whether it succeeded, failed, or was abandoned, this page shows the live log file, the details, and the history for each step or task. You can download the log file for each task, or download all of the log files as a zip file.





If you don't want to see the live log output, hide it using the **Details pane** drop-down list at the top right of the page. When the details pane is hidden, you can sort the list in ascending or descending order based on the values in the columns by clicking the column headings (the ▼ icon shows the current sorting column).

The screenshot shows the 'FabrikamDev / Release-1' page. At the top, there are tabs: Summary, Environments, Artifacts, Configuration, General, Commits, and Work. To the right of these tabs is a 'View All' button and a 'Details pane' dropdown menu. The 'Details pane' dropdown is open, showing 'On' and 'Off' options. Below the tabs, there are icons for refresh, add, and a list icon, followed by buttons: 'Abandon', '+ Deploy', and 'Download all logs as zip'. Below this is a table with columns: Step, Start Date, Duration, and Details. The table lists several steps: Pre-deployment approval, Deploy, Download artifacts, Deploy Website to Azure, Run Tests, and Post-deployment approval. The 'Details pane' dropdown is highlighted with an orange box.

As the list of environments and tasks grows, including the history for earlier releases, it may be more difficult to find individual items. Filter the list to show just approvals or just tasks using the **View** drop-down list at the top right of the page.

The screenshot shows the 'FabrikamDev / Release\*: Release-6' page. At the top, there are tabs: Summary, Environments, Artifacts, Configuration, General, and Commits. To the right of these tabs is a 'View' dropdown menu and a 'Details pane' dropdown menu. The 'View' dropdown is open, showing 'All', 'Approvals', and 'Tasks' options. Below the tabs, there are icons for refresh, add, and a list icon, followed by buttons: 'Abandon', '+ Deploy', and 'Download all logs as zip'. Below this is a table with columns: Step, Start Date, Duration, and Details. The table lists several steps: Pre-deployment approval and Post-deployment approval. The 'View' dropdown is highlighted with an orange box.

#### 6.2.5.5 Redeploying after failure

If a release fails or is canceled, you can redeploy it. For example, if an approver canceled the release because an environment was temporary offline, you can redeploy it when the environment is available again. If a release fails, you can redeployed it after you fix the problem that caused the failure - perhaps there was a connectivity failure or a server was temporary offline.

To redeploy a release, open the **Summary** page, use the ellipses (...) to open the shortcut menu for the environment you want to redeploy to, and choose **Redeploy**.

FabrikamDev / Release-3

**Summary** Environments Artifacts Configuration General Commits Work items Logs

🔄 | 💾 Save Abandon | + Deploy ▾

Details		Work items
Release-3		No associated work items found.
Manually created by Alex Homer 10 hours ago		
🏠 FabrikamDev / 20160127.1 (Build) 🐦 refs/heads/master		
Environments		Test results
		No test results available for this release.

Environment	Actions	Deployment Status	Started	Duration
Dev	⋮	REJECTED	10 hours ago	00:00:28

**Issues**

No issues reported in this release.

**Actions:** Deploy, Cancel, Redeploy

You can also use the **Deploy** list in the **Summary** page to start a new deployment for a release...

FabrikamDev / Release-6

**Summary** Environments Artifacts Configuration General Commits Work items Logs

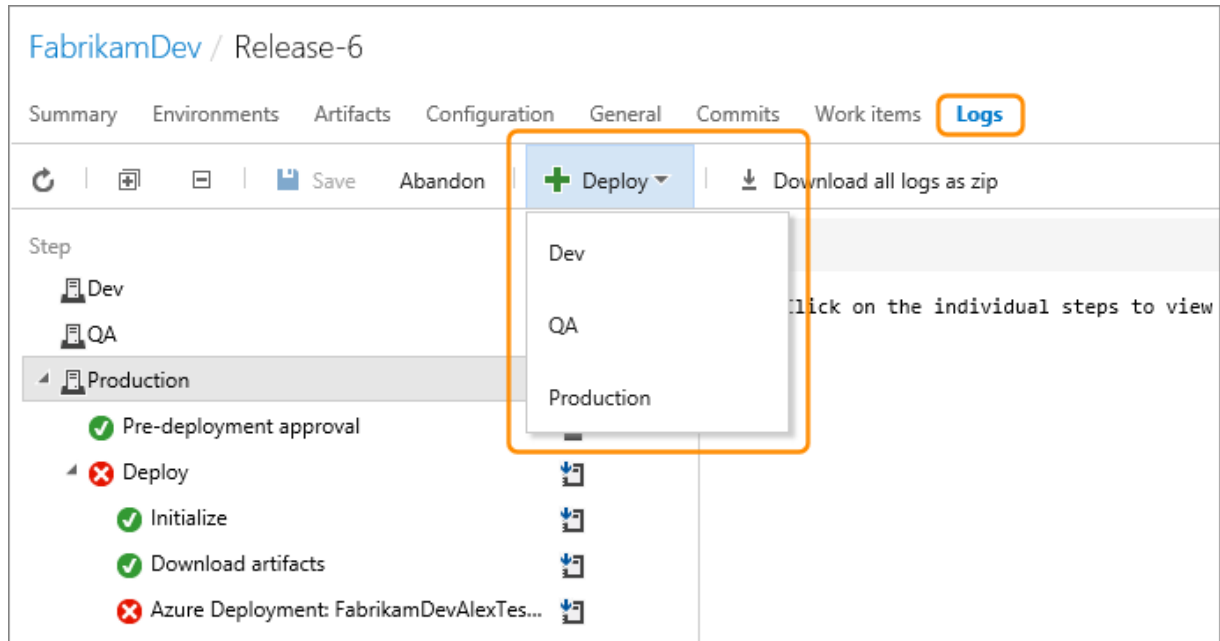
🔄 | 💾 Save Abandon | + Deploy ▾

Details		Work items
Release-6		No associated work item
Manually created by Mateo Escobedo		
🏠 FabrikamDev / 20160127.1 (Build)		
Environments		Test results
		No test results available

Environment	Actions	Deployment Status	Started	Duration
Dev	⋮	NOT DEPLOY...		
QA	⋮	NOT DEPLOY...		
Production	⋮	REJECTED	less than a minu...	00:00:28

**Deploy List:** Dev, QA, Production

...or the **Deploy** list in the **Logs** page.



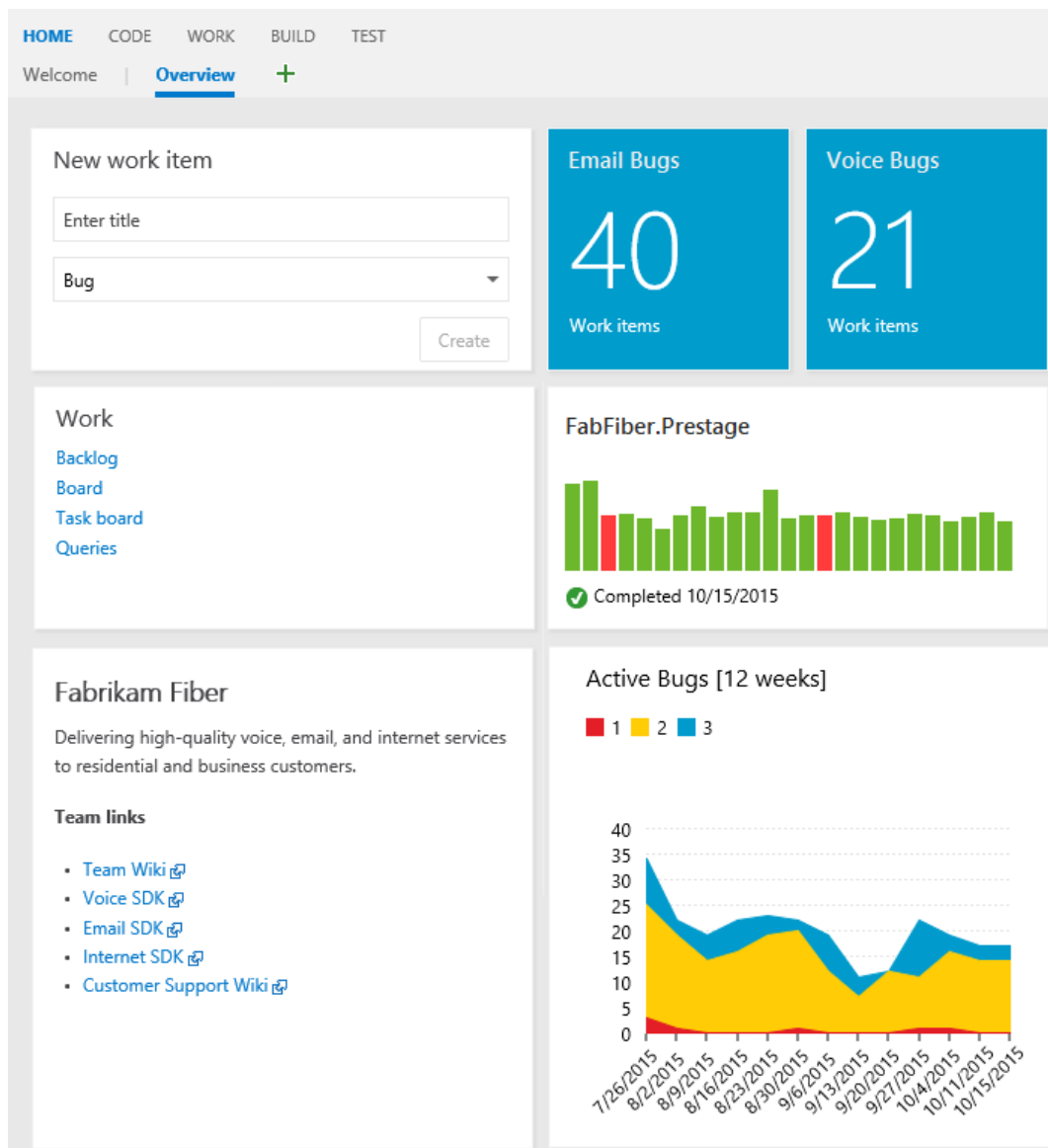
If you need to change the release definition before you redeploy, you must **create a new release** from the updated release definition.

# 7 Reporting

## 7.1 Dashboards

Share progress and status with your team using configurable team dashboards. Dashboards provide easy-to-read, easy access, real-time information. At a glance, you can make informed decisions without having to drill down into other parts of your team project site.

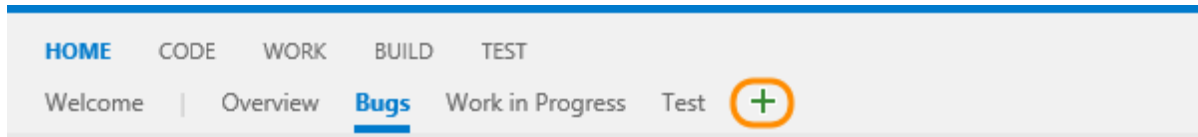
The Overview page provides access to a default team dashboard which you can customize by adding, removing, or rearranging the tiles. Each tile corresponds to a widget that provides access to one or more features or functions.




Anyone with access to the team project, including stakeholders, can view dashboards. However, only team admins can add or modify dashboards.

### 7.1.1 Add and name your dashboard

From the dashboards tab, click the  and enter a dashboard name.



If you don't see the , then you're not a team admin for the currently selected team. Either switch the context to your team, or request you be added as a team admin.

With the dashboard selected, you can add a widget to the dashboard. Or, you can add a chart to a team dashboard from the Work, Build, or Test hubs.


### 7.1.2 Add a widget to your dashboard



Click  to modify a dashboard. Click  to add a widget to the dashboard.



The widget catalog describes all the available widgets, many of which are scoped to the selected team context.

After you add the widget, you may need to configure it. For example, to configure the Query tile widget, click the  to open the configuration dialog.



And then select the query and optionally the green and red flag limits.

Configuration

General

Name
Active Bugs

Data

Query
Active Bugs

Visualization

Background color

☒
If number of work items
<=
30

☒
If number of work items
>
80

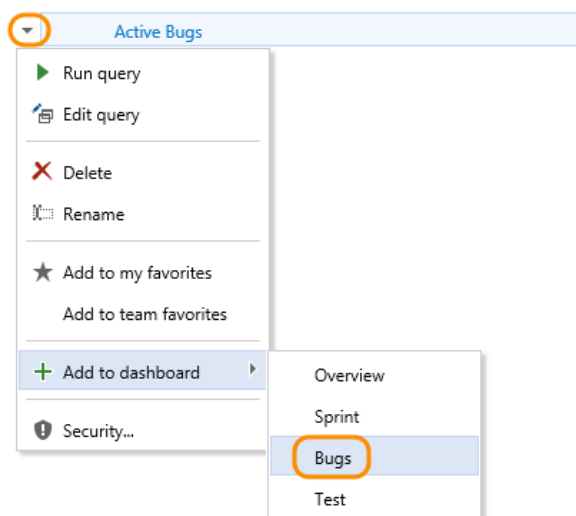
Save
Cancel

### 7.1.3 Add an item or a chart to your dashboard

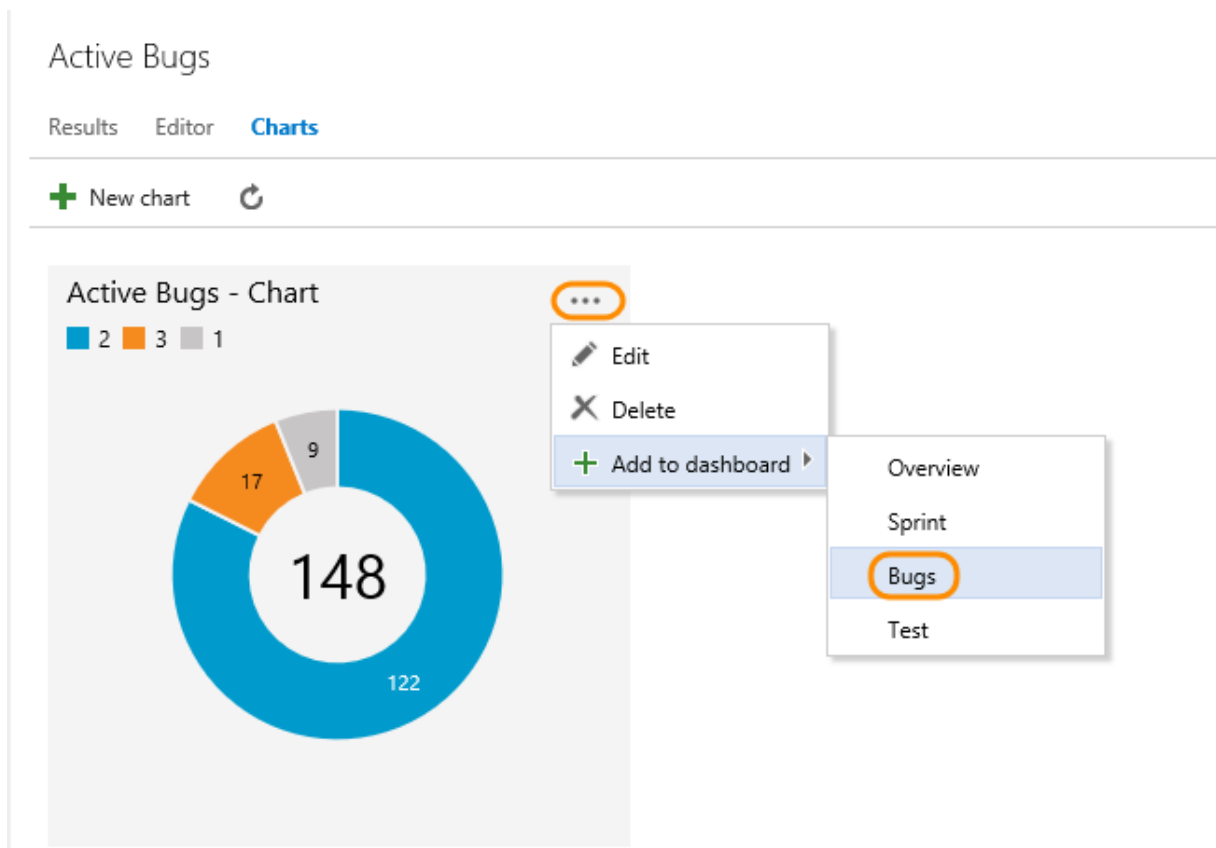
You add an item to a dashboard from the code, work, and build pages.

First, make sure you have the team context selected from the Queries page to which you want to add a query or chart.

For example, select the context menu of a query that you want to add to the dashboard. This is the same as adding a query tile widget.




And, you can add a chart to a team dashboard in a similar way.



#### 7.1.4 Reorder or remove widgets on the dashboard




Click  to modify your dashboard. You can then drag tiles to reorder their sequence on the dashboard.



To remove a widget, click the widget's .


Just as you have to be a team or project admin to add items to a dashboard, you must have admin permissions to remove items.




When you're finished with your changes, click  to exit dashboard editing.


## 7.2 Manage dashboards


You can rename, reorder, or delete a dashboard. Also, you can enable auto-refresh, and the dashboard will automatically update every 5 minutes.


1. To change the sequence in which dashboards are listed, click the  gear icon to open manage dashboards.


2. Drag and drop the dashboards into the sequence you want them to appear.


MANAGE DASHBOARDS 








Auto-refresh dashboard ☐ 




Auto-refresh dashboard ☒ 



Auto-refresh dashboard ☒ 



Auto-refresh dashboard ☒ 

Done

Cancel

3. Click  to delete a dashboard and then click Done.



# End of Book 😊