Testing Automation

How to View This Presentation

- Switch to the **Notes Page** view:
  - Click **View** on the ribbon, and select **Notes Page**
  - Use the Page Up or Page Down keys to navigate
  - Zoom in or out as needed
- In the **Notes Page** view, you can:
  - Read any supporting text, now or after the delivery
- Add your own notes
- Take the presentation files home with you

Introduction and Logistics

- Your trainer
- You:
  - Your role
  - Your company
  - Your experience in this technology area
  - Your goals for this workshop
- Start and end times
- Facilities (restrooms and smoking)
- Meals
- Devices such as computers, phones, tablets, and so on
- Please set your mobile phones to vibrate
- What is on your desk?

Module: Continuous Testing in DevOps

Module Overview

Overview

- Continuous Testing using Azure Pipelines
- Running Tests
- Reviewing Test Results

Module: Continuous Testing in DevOps

Lesson 1: Continuous Testing using Azure Pipelines

Overview

- Testing in a Continuous Integration (CI) / Continuous Deployment (CD) Scenario
- Azure Pipelines – Tasks
- Running Tests in Parallel
- Running Tests in Parallel using Visual Studio Test Task and Jobs
- Running Tests in Parallel in Build Pipelines Using Jobs
- Running Tests in Parallel in Release Pipelines Using Jobs
- How Visual Studio Test Task Runs in Jobs
- Massively Parallel Testing

Testing in a CI / CD Scenario

- Run **unit tests** in the CI pipeline
- Run **functional tests** in the early stages of the CD pipeline (Example: after your app is deployed to a QA environment).
  - Selenium (for web apps) and Coded UI (Visual Studio 2019 is the last version that supports Coded UI test type).
- Run **load tests** after the app is deployed to staging and production, after it passes all functional tests

Azure Pipelines - Tasks

- **Visual Studio Test Agent Deployment Task**: Deprecated in Azure Pipelines
- **Run Functional Tests Task**: Deprecated in Azure Pipelines
- **Visual Studio Test Task (version 2.x or higher)**:
  - Run unit and functional tests (Selenium, Appium, Coded UI, and more) using the Visual Studio Test Runner in a build or release pipeline
  - Test frameworks that have a Visual Studio test adapter, such as xUnit, NUnit, Chutzpah, can also be executed
  - Satisfies the vstest demand in two ways:
    - Visual Studio is installed on the agent machine
    - By using the Visual Studio Test Platform Installer task in the pipeline definition
- **Visual Studio Test Platform Installer Task**:
  - Acquires the Microsoft test platform from nuget.org or a specified feed, and adds it to the tools cache in a build or release pipeline
  - Satisfies the 'vstest' demand and a subsequent Visual Studio Test task in a build or release pipeline can run without requiring a full Visual Studio install on the agent machine

Running Tests in Parallel

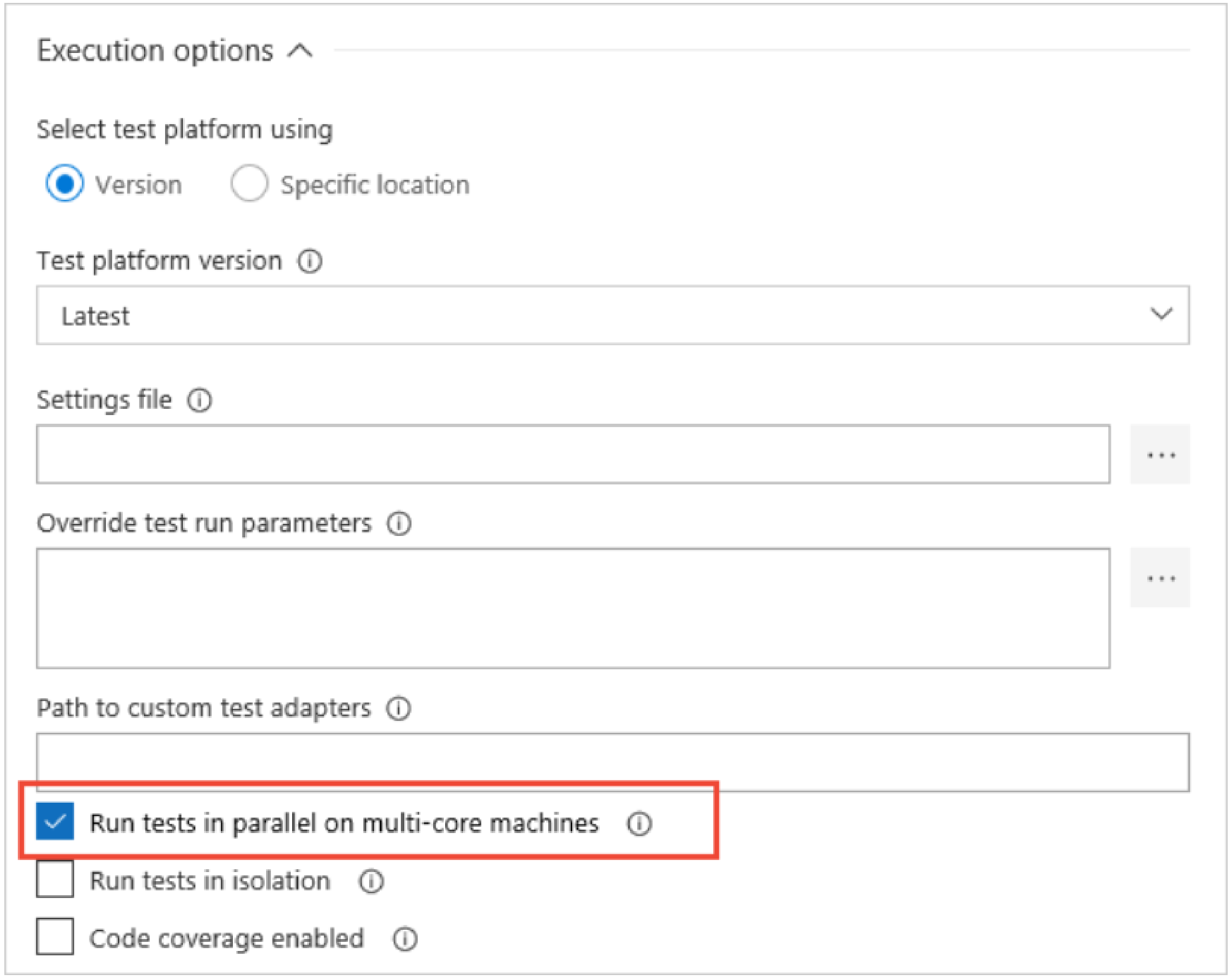
- The Visual Studio Test Platform (VSTest) supports running tests in parallel.
- Parallel test execution is available:
  - To all frameworks and within the IDE, the command line (CLI), and in **Azure Pipelines**.
  - Within the IDE from all launch points (Test Explorer, CodeLens, various Run commands, and more).
- Parallel test execution:
  - Composes with test adapters and frameworks that already support parallel execution such as MSTest, NUnit, and xUnit.net.
  - OFF by default, users must explicitly opt-in
  - Supported at assembly level
- Parallel Test Execution is **not** supported in the following cases:
  - If the test run is configured using a **.testsettings** file.
  - For test code targeting Phone, Store, UWP app platforms.

Running Tests in Parallel (continued)

- Partition tests in terms of a taxonomy as follows:
  1. Pure unit tests (typically these can run in parallel)
  2. Functional tests that can run in parallel with some modifications
  3. Functional tests that cannot be modified to run in parallel
- Gradually evolve the partitioning as follows:
  1. Run the tests in parallel, see which tests fail, and classify them as above
  2. Fix tests in category 2 so that they can run in parallel
  3. Move tests in category 3 into a separate test run where parallel is OFF by default

Running Tests in Parallel (continued)

- Enable parallel test execution in **Azure Pipelines** by setting the **Run Tests in Parallel...** checkbox in the settings for the **Visual Studio Test** task.
- When parallel test execution is enabled, the value of **MaxCpuCount** is set to 0 (zero) to use all the available free cores.



Execution options

Select test platform using

[✓ Version] Specific location

Test platform version ⓘ

Latest

Settings file ⓘ

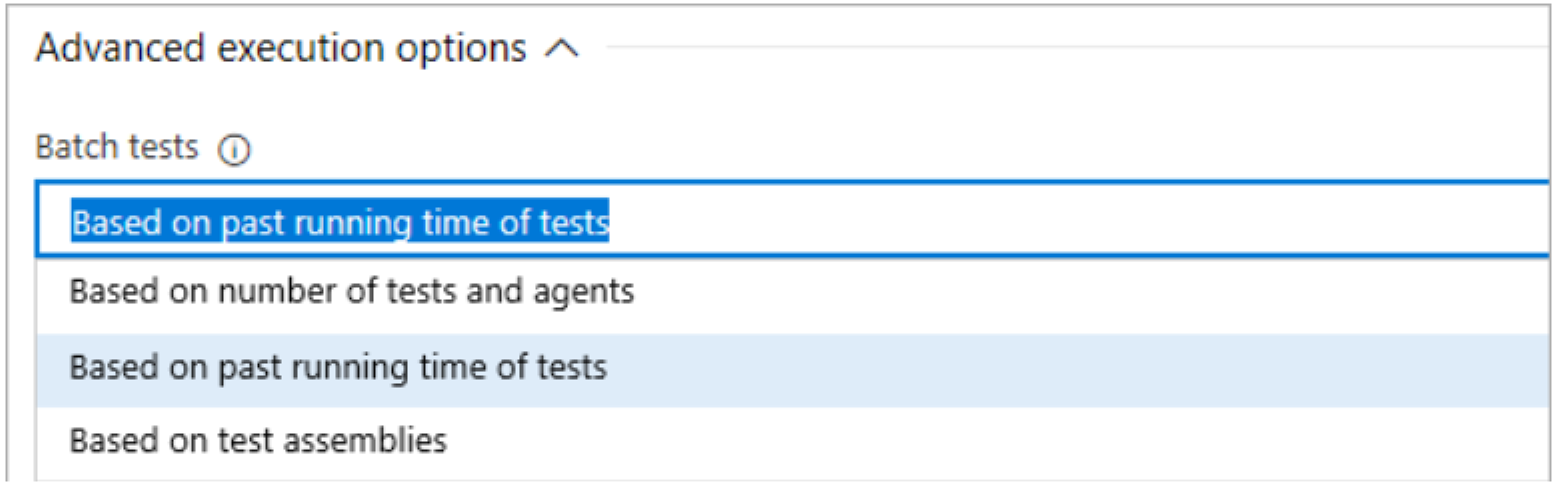
Override test run parameters ⓘ

Path to custom test adapters ⓘ

- [x] Run tests in parallel on multi-core machines ⓘ
- [ ] Run tests in isolation ⓘ
- [ ] Code coverage enabled ⓘ

Running Tests in Parallel using Visual Studio Test Task and Jobs

- To run multiple jobs in parallel, you must configure multiple agents. You also need sufficient parallel jobs.
- When a pipeline job that contains the VSTest task is configured to run on multiple agents in parallel, it automatically detects that multiple agents are involved and creates test slices that can be run in parallel across these agents.
- The task can be configured based on:
  - the number of tests and agents,
  - the previous test running times, or
  - the location of tests in assemblies.


Advanced execution options

Batch tests

Based on past running time of tests

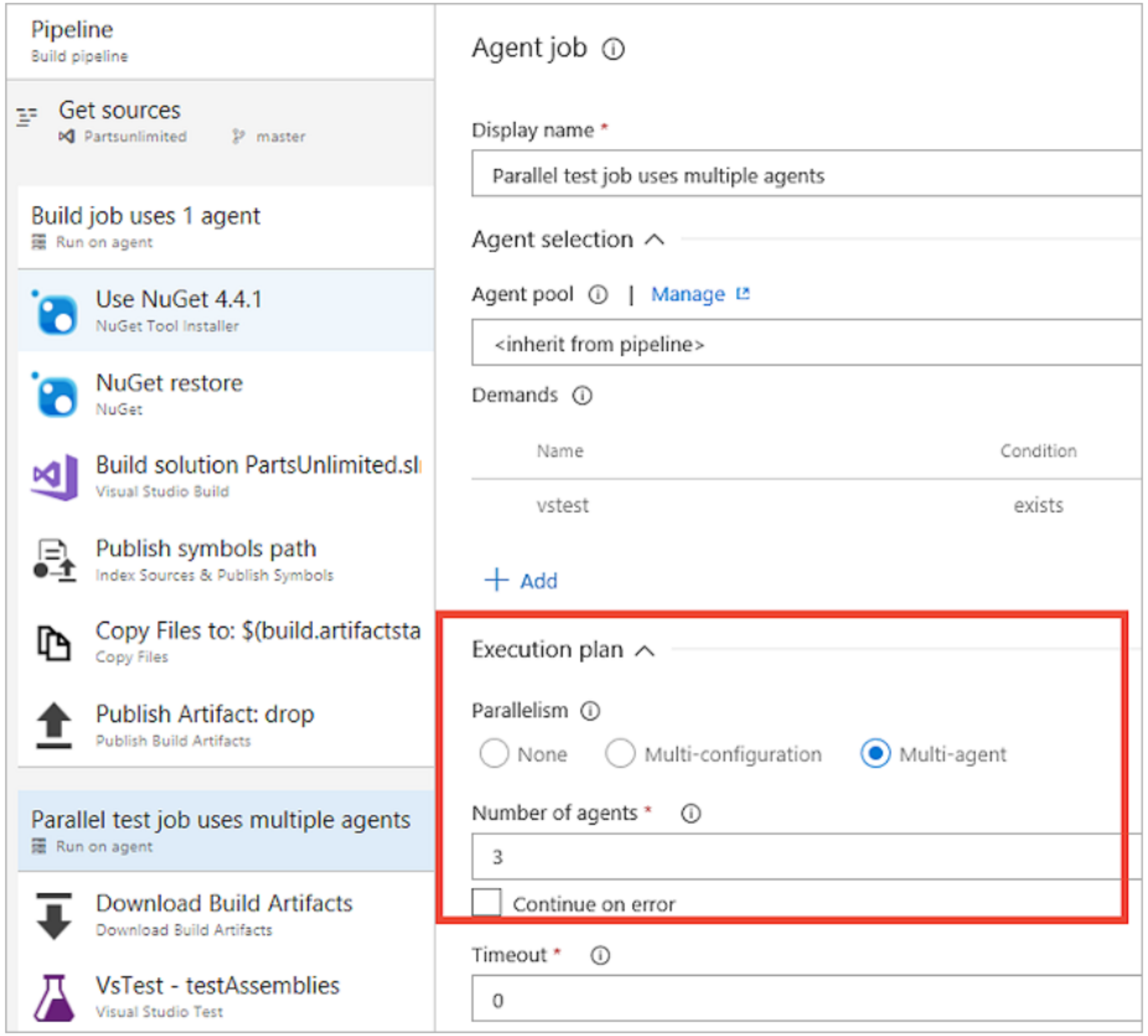
Based on number of tests and agents

Based on past running time of tests

Based on test assemblies

Running Tests in Parallel in Build Pipelines Using Jobs

- **Job 1** : Build your projects and publish build artifacts using the default job settings (single agent, no parallel jobs)
- **Job 2** : Add an agent job, and configure it to use **multiple agents in parallel**. Add the following tasks:
  - **Download Build Artifacts**: This step is the link between the build job and the test job and is necessary to ensure that the binaries generated in the build job are available on the agents used by the test job to run tests.
  - **Visual Studio Test:** Configure it to use the required slicing strategy.



Pipeline
Build pipeline

Get sources
Partsunlimited   master

Build job uses 1 agent
Run on agent

Use NuGet 4.4.1
NuGet Tool Installer

NuGet restore
NuGet

Build solution PartsUnlimited.sl
Visual Studio Build

Publish symbols path
Index Sources & Publish Symbols

Copy Files to: $(build.artifactsta
Copy Files

Publish Artifact: drop
Publish Build Artifacts

Parallel test job uses multiple agents
Run on agent

Download Build Artifacts
Download Build Artifacts

VsTest - testAssemblies
Visual Studio Test

Agent job

Display name *
Parallel test job uses multiple agents

Agent selection

Agent pool | Manage
<inherit from pipeline>

Demands

Name	Condition
vstest	exists

+ Add

Execution plan

Parallelism
None   Multi-configuration   [✓ Multi-agent]

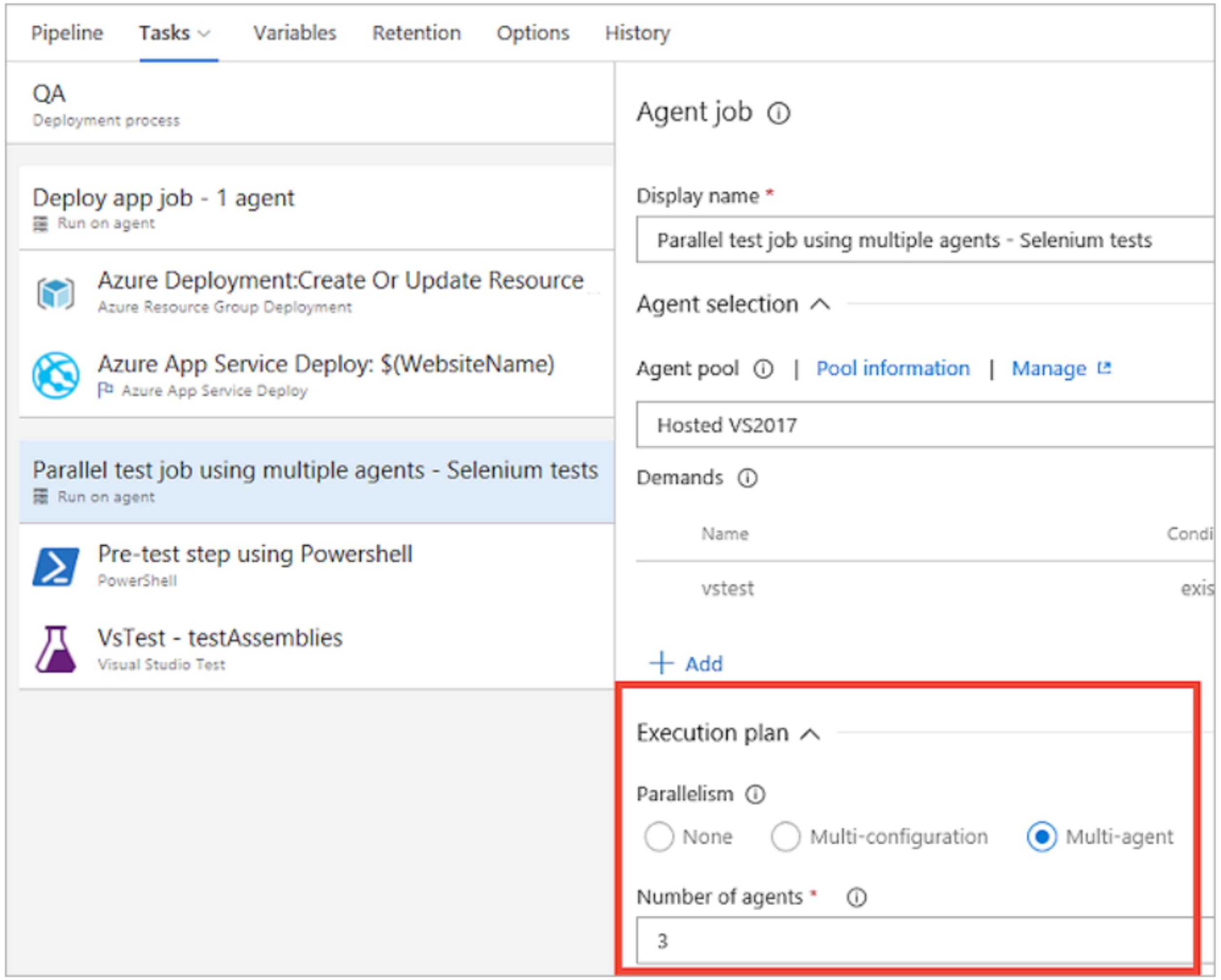
Number of agents *
3

Continue on error

Timeout *
0

Running Tests in Parallel in Release Pipelines Using Jobs

- **Job 1** : Deploy your application using the default job settings (single agent, no parallel jobs)
- **Job 2** : Add an agent job, and configure it to use **multiple agents in parallel**. Add the following tasks:
    - **Additional tasks** that must run before the Visual Studio test task is run. For example, run a PowerShell script to set up any data required by your tests.
    - **Visual Studio Test**: Configure it to use the required slicing strategy.
- Jobs in release pipelines download all artifacts linked to the release pipeline by default. You can configure the job to download only the test artifacts required by the job.



QA
Deployment process

Deploy app job - 1 agent
Run on agent

Azure Deployment:Create Or Update Resource
Azure Resource Group Deployment

Azure App Service Deploy: $(WebsiteName)
Azure App Service Deploy

Parallel test job using multiple agents - Selenium tests
Run on agent

Pre-test step using Powershell
PowerShell

VsTest - testAssemblies
Visual Studio Test

Agent job

Display name *: Parallel test job using multiple agents - Selenium tests

Agent selection

Agent pool | Pool information | Manage

Hosted VS2017

Demands

Name	Condi
vstest	exis

+ Add

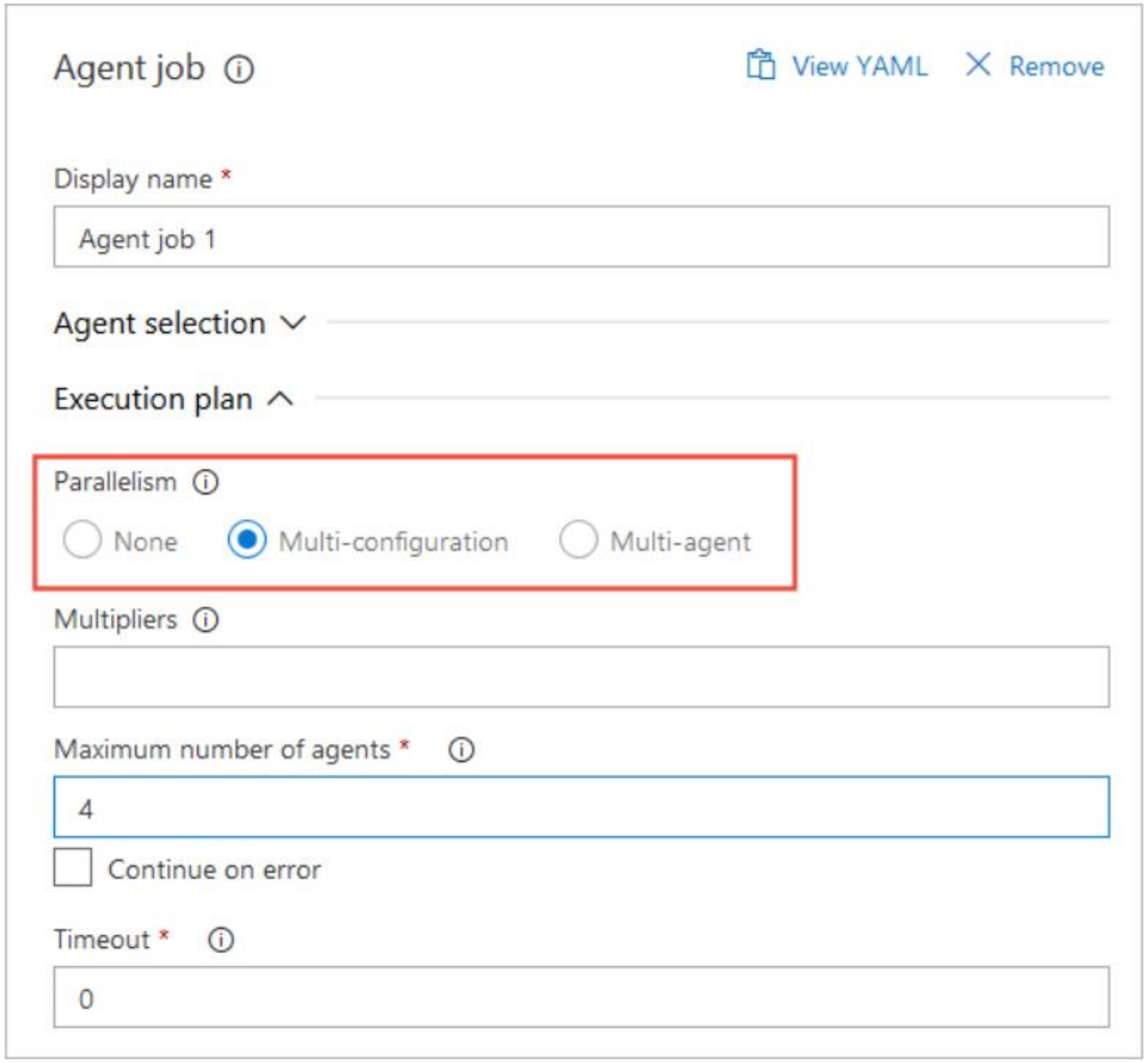
Execution plan

Parallelism: None [ ] Multi-configuration [ ] [✓ Multi-agent]

Number of agents *: 3

How Visual Studio Test Task Runs in Jobs

- **No Parallelism**
- **Multiple Executions**
- **Multiple Agents**



Agent job ⓘ

View YAML Remove

Display name *

Agent job 1

Agent selection

Execution plan

Parallelism ⓘ

None [✓ Multi-configuration] Multi-agent

Multipliers ⓘ

Maximum number of agents * ⓘ

4

☐ Continue on error

Timeout * ⓘ

0

Massively Parallel Testing

- When parallel jobs are used in a pipeline, it employs multiple machines (agents) to run each job in parallel.
- Test frameworks and runners also provide the capability to run tests in parallel on a single machine, typically by creating multiple processes or threads that are run in parallel.
- In the context of the Visual Studio Test task, parallelism can be combined in the following ways:
    - Parallelism offered by test frameworks
    - Parallelism offered by the Visual Studio Test Platform (vstest.console.exe)
    - Parallelism offered by the Visual Studio Test (VSTest) task

Demo 1: Continuous Testing using Azure Pipelines

Lesson Knowledge Check

1. True/False : To be able to use the VSTest task, you need to deploy a Visual Studio Test Agent.
2. What are the parallel testing options offered by the VSTest task?
3. In how many ways can the VSTest task create test slices?

Lesson Summary

- In this lesson, you learned about:
  - Testing in a CI / CD Scenario
  - Azure Pipelines – Tasks
  - Running Tests in Parallel
  - Running Tests in Parallel using Visual Studio Test Task and Jobs
  - Running Tests in Parallel in Build Pipelines Using Jobs
  - Running Tests in Parallel in Release Pipelines Using Jobs
  - How Visual Studio Test Task Runs in Jobs
  - Massively Parallel Testing

Module: Continuous Testing in DevOps

Lesson 2: Running Tests

Overview

- Run Unit Tests with Builds
- Publish Code Coverage Results
- Test Impact Analysis
- Run UI Tests with Releases
- UI Testing Considerations

Run Unit Tests with Builds

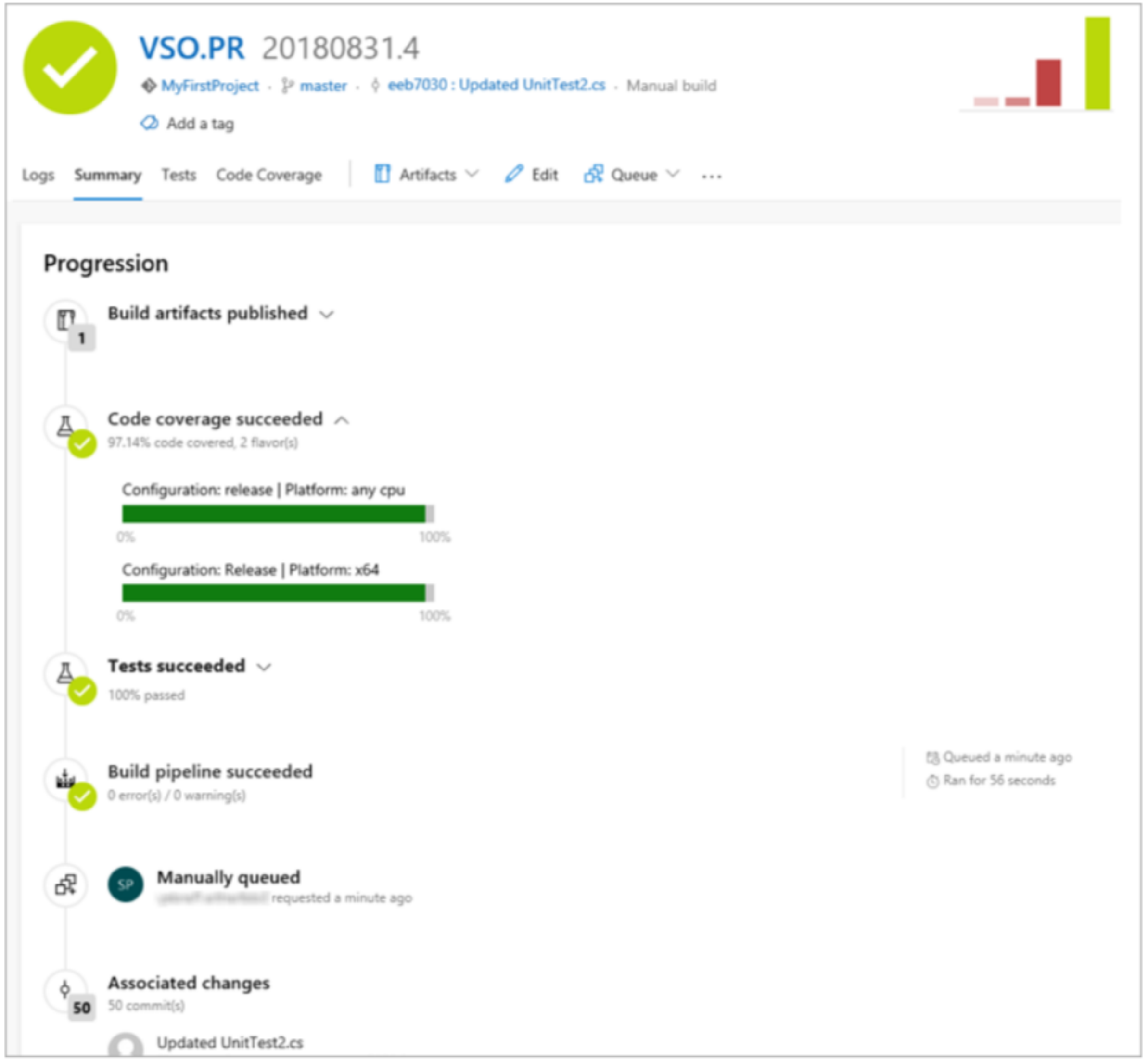
- Make sure that your app builds after every check-in by using test automation in Azure Pipelines.
- Find problems earlier by running tests automatically with each build.
- When your build is done, review your test results to start resolving the problems that you find.

Run Unit Tests with Builds (continued)

- In order to run unit tests with your builds:
  - **Check-in / Push** your solution to Azure Repos. Include your test projects.
  - **Create a build pipeline**.
    - Your build pipeline must include a test task that runs unit tests.
    - For example, if you're building a Visual Studio solution in Azure Pipelines, your build pipeline should include a **Visual Studio Test task**. After your build starts, this task automatically runs all the unit tests in your solution - on the same build machine.
    - Edit the test task. For example, for the Visual Studio Test task, you can add filter criteria to run specific tests, enable code coverage, run tests from other unit test frameworks, and so on.
  - **Start the build**.
  - After the build finishes, you can **review the test results** to resolve any problems that happened. Go to the build summary and open the Tests page.

Publish Code Coverage Results

- Code coverage helps you determine the proportion of your project's code that is actually being tested by tests such as unit tests.
- You can **publish code coverage results** in a build pipeline by
  - Using the "Publish Code Coverage Results" task (or)
  - Using built-in tasks such as "Visual Studio Test" that provide the option to publish code coverage data to the pipeline.
- The code coverage summary can be viewed in the **build timeline view**. The summary shows the overall percentage of line coverage.
- The code coverage artifacts published during the build can be viewed under the **Build artifacts published** milestone in the timeline view.


VSO.PR 20180831.4

MyFirstProject · master · eeb7030 : Updated UnitTest2.cs · Manual build

Add a tag

Progression

Build artifacts published
1

Code coverage succeeded
97.14% code covered, 2 flavor(s)

Configuration: release | Platform: any cpu
0% 100%

Configuration: Release | Platform: x64
0% 100%

Tests succeeded
100% passed

Build pipeline succeeded
0 error(s) / 0 warning(s)

Queued a minute ago
Ran for 56 seconds

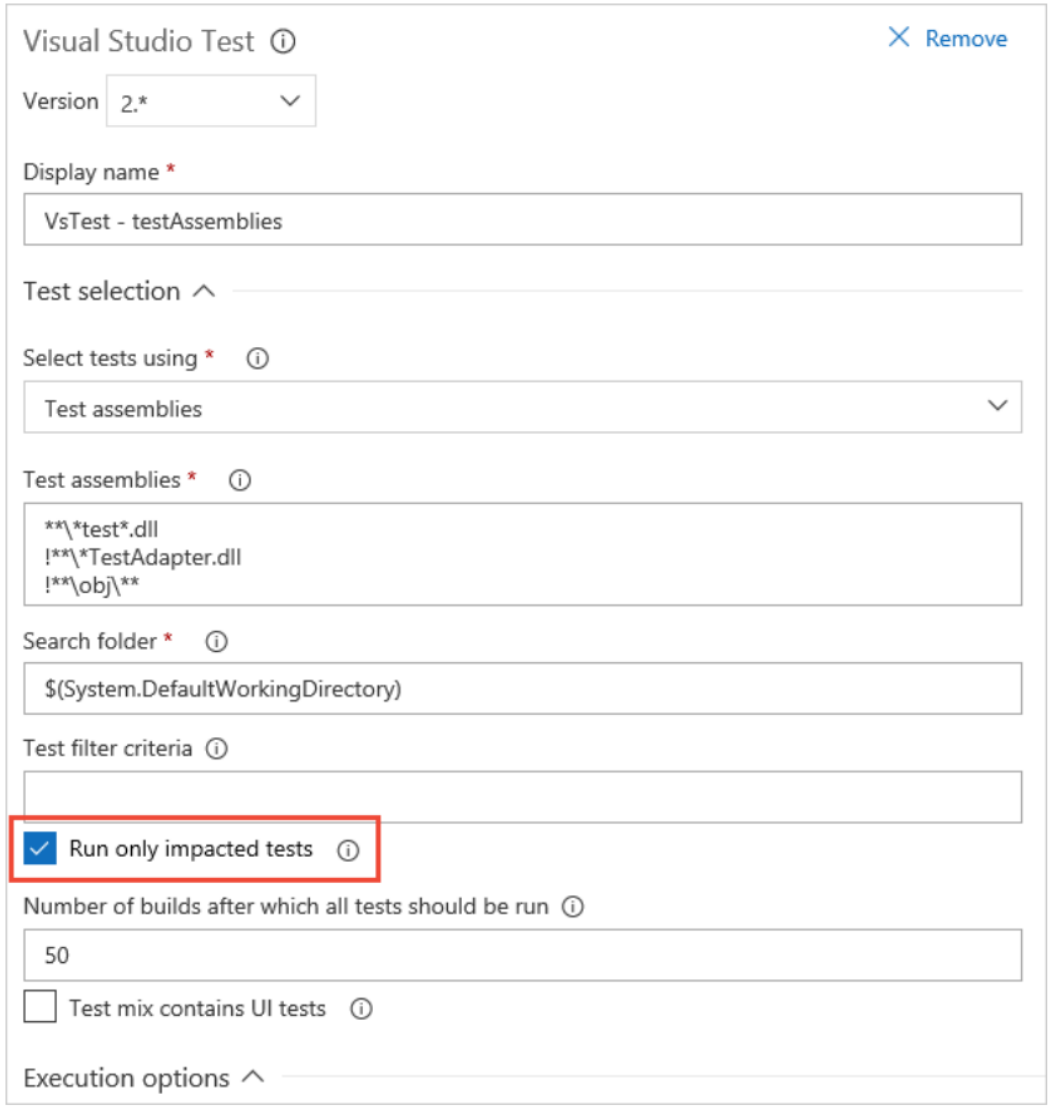
Manually queued
requested a minute ago

Associated changes
50
50 commit(s)

Updated UnitTest2.cs

Test Impact Analysis

- Enable Test Impact Analysis (TIA) when using the Visual Studio Test task in a build pipeline
  - For a given code commit entering the CI/CD pipeline, TIA will select and run only the relevant tests required to validate that commit.



Version: 2.*

Display name *: VsTest - testAssemblies

Test selection

Select tests using *: Test assemblies

Test assemblies *:

```
**\*test*.dll
!**\*TestAdapter.dll
!**\obj\**
```

Search folder *: $(System.DefaultWorkingDirectory)

Test filter criteria:

[✓] Run only impacted tests

Number of builds after which all tests should be run: 50

[ ] Test mix contains UI tests

Execution options

Run UI Tests with Releases

- Performing user interface (UI) testing as part of the release pipeline is a great way of detecting unexpected changes.

Run UI Tests with Releases (continued)

- In order to run Selenium tests with your releases:
  - **Create your test project**
    - Use the Unit Test Project template in Visual Studio.
    - Add the Selenium and browser driver references used by the browser to execute the tests.
    - Create your tests.
    - Run the Selenium test locally using Test Explorer and check that it works.
  - **Create a build pipeline** to build your application and your test project.
  - **Decide** how you will deploy and test your app
    - You can deploy and test your app using either the Microsoft-hosted agent in Azure, or a self-hosted agent that you install on the target servers.
      - When using the **Microsoft-hosted agent**, you should use the Selenium web drivers that are pre-installed on the Windows agents.
      - When using a **self-hosted agent** that you deploy on your target servers, agents must be configured to run interactively with auto-logon enabled.
  - **Create a release pipeline**.
    - Add the build artifacts.
    - Add the **Visual Studio Test Platform Installer** task if the target machines that host the agents do not have Visual Studio installed.
    - Add the **Visual Studio Test** task.
  - **Start the release**.
  - After the release finishes, you can **review the test results**.

UI Testing Considerations

- When running Selenium tests for a web app, you can launch the browser in two ways:
  - **Headless mode**
  - **Visible UI mode**
- If you are running UI tests for a desktop application, such as Appium tests using WinAppDriver or Coded UI tests, a special configuration of the agents is required.

Demo 2: Running Tests

Lesson Knowledge Check

1. How can you publish code coverage results in a build pipeline?
2. What are the steps needed to run UI tests in a release pipeline?
3. When running Selenium tests for a web app, in what ways can you launch the browser?

Lesson Summary

- In this lesson, you learned about:
    - Run Unit Tests with Builds
    - Publish Code Coverage Results
    - Test Impact Analysis
    - Run UI Tests with Releases
    - UI Testing Considerations

Module: Continuous Testing in DevOps

Lesson 3: Reviewing Test Results

Overview

- Glossary
- Review Test Results
- View Test Results in Build
- View Test Results in Release
- Tests Tab
- Surface Test Results in the Tests Tab
- Surface Test Information Beyond the Tests Tab
- Test Analytics
- Test Result Trend (Advanced) Widget
- Trace Test Requirements
- Flaky Test Management

Glossary

Term	**Definition**
Duration	Time elapsed in execution of a **test**, **test run**, or **entire test execution** in a build or release pipeline.
Owner	Owner of a **test** or **test run**. The test owner is typically specified as an attribute in the test code.
Failing build	Reference to the **build** having the first occurrence of consecutive failures of a test case.
Failing release	Reference to the **release** having the first occurrence of consecutive failures of a test case.
Outcome	There are 13 possible outcomes for a test result: Aborted, Blocked, Error, Failed, Inconclusive, None, Not applicable, Not executed, Not impacted, Passed, Paused, Timeout, Unspecified and Warning
Flaky test	A test with non-deterministic behavior. For example, the test may result in different outcomes for the same configuration, code, or inputs.

Glossary (continued)

Term	Definition
Filter	Mechanism to search for the test results within the result set, using the available attributes.
Grouping	An aid to organizing the test results view based on available attributes such as **Requirement**, **Test files**, **Priority**, and more. Both test report and test analytics provide support for grouping test results.
Pass percentage	Measure of the success of test outcome for a single instance of execution or over a period of time.
Priority	Specifies the degree of importance or criticality of a test. Priority is typically specified as an attribute in the test code.
Test analytics	A view of the historical test data to provide meaningful insights.
Test case	Uniquely identifies a single test within the specified branch.
Test files	Group tests based on the way they are packaged; such as files, DLLs, or other formats.

Glossary (continued)

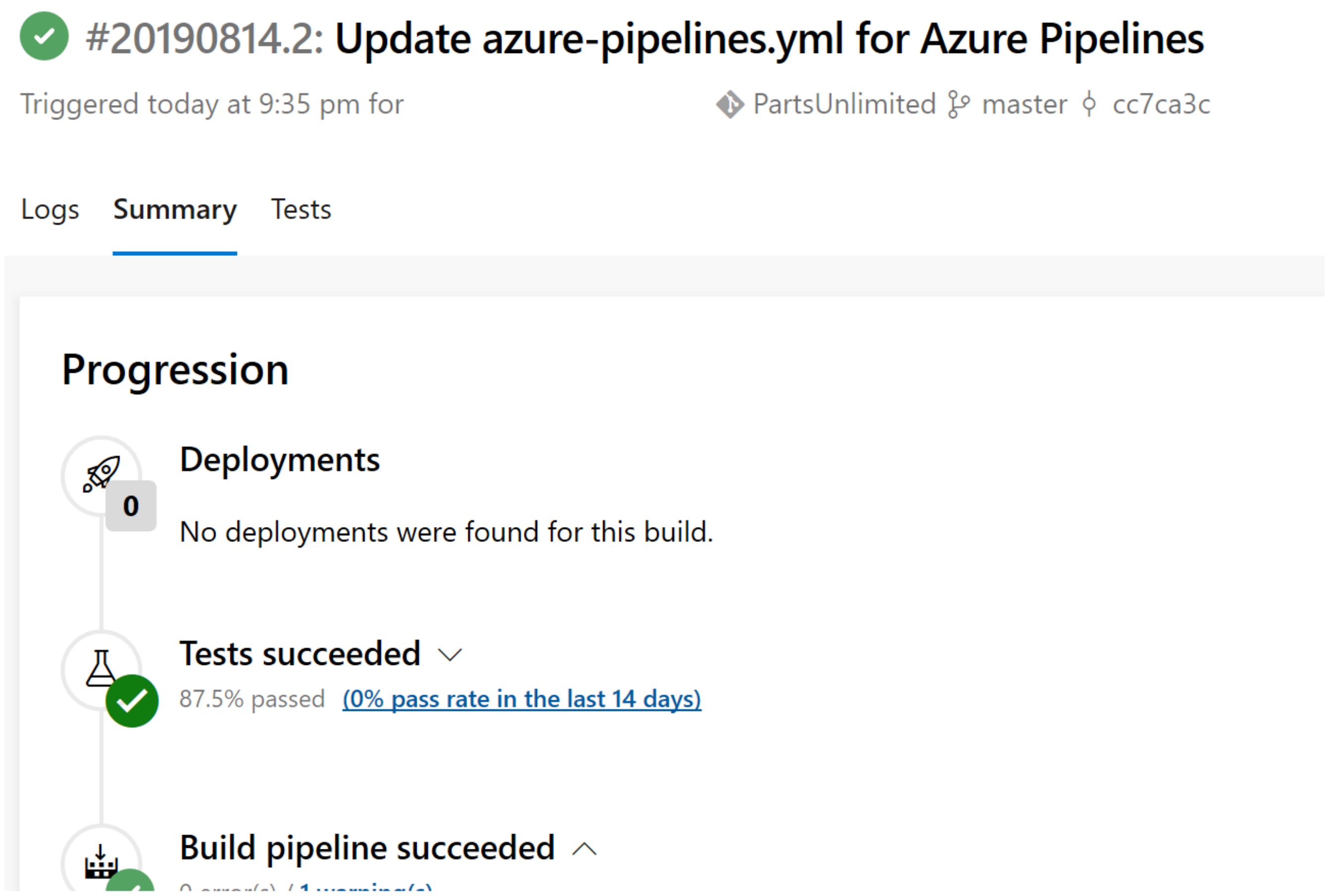
Term	Definition
Test report	A view of single instance of test execution in the pipeline that contains details of status and help for troubleshooting, traceability, and more.
Test result	Single instance of execution of a test case with a specific outcome and details.
Test run	Logical grouping of test results based on: - Test executed using built-in tasks - Results published using Publish Test Results task - Tests results published using API(s)
Traceability	Ability to trace forward or backward to a requirement, bug, or source code from a test result.

Review Test Results

- Automated tests can be configured to run as part of a build or release for various languages.
- Test reports provide an effective and consistent way to view the tests results executed using different test frameworks, in order to measure pipeline quality, review traceability, troubleshoot failures and drive failure ownership.
- Published test results can be viewed in the **Tests** tab in a build or release summary.

View Test Results in Build

- The build summary provides a timeline view of the key steps executed in the build. If tests were executed and reported as part of the build, a test milestone appears in the timeline view.
- The test milestone provides a summary of the test results as a measure of **pass percentage** along with indicators for **failures** and **aborts** if these exist.



#20190814.2: Update azure-pipelines.yml for Azure Pipelines

Triggered today at 9:35 pm for PartsUnlimited master cc7ca3c

Progression

Deployments
0
No deployments were found for this build.

Tests succeeded
87.5% passed (0% pass rate in the last 14 days)

Build pipeline succeeded

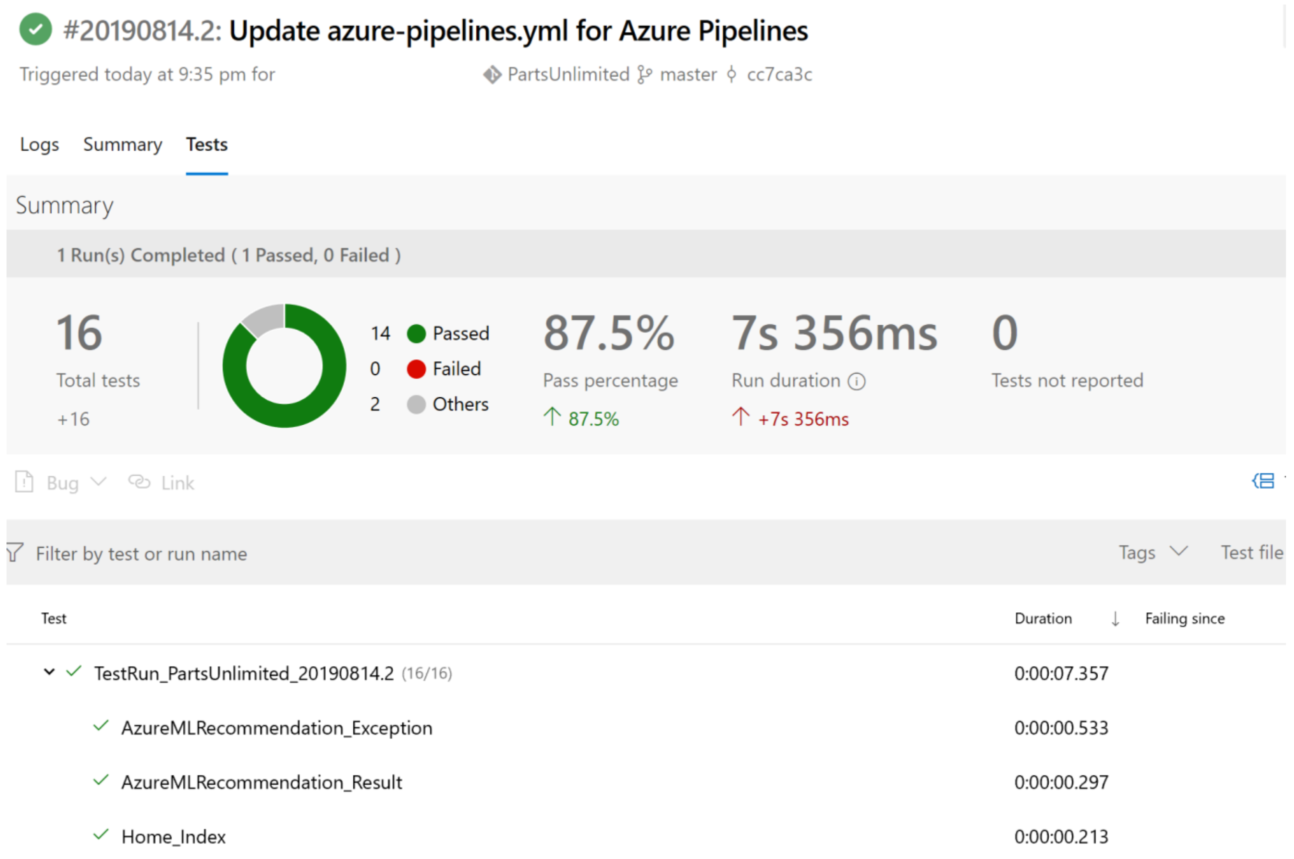
Microsoft Confidential

View Test Results in Release

- In the pipeline view, you can see all the stages and associated tests.
- The view provides a summary of the test results as a measure of **pass percentage** along with indicators for **failures** and **aborts** if these exist.
- These indicators are same as in the build timeline view, giving a consistent experience across build and release.

Tests Tab

- Both the build and release summaries provide details of test execution.
- **Tests** tab has the following sections:
  - **Summary**
  - **Results**
  - **Details**
- Select any test run or result to view the details pane that displays additional information required for troubleshooting such as:
  - Errors, stack trace, attachments, work items, historical trend, and more.



#20190814.2: Update azure-pipelines.yml for Azure Pipelines

Triggered today at 9:35 pm for PartsUnlimited master cc7ca3c

Logs Summary Tests

Summary

1 Run(s) Completed (1 Passed, 0 Failed)

16 Total tests +16

14 ● Passed
0 ● Failed
2 ● Others

87.5% Pass percentage ↑ 87.5%

7s 356ms Run duration ↑ +7s 356ms

0 Tests not reported

Bug Link

Filter by test or run name Tags Test file

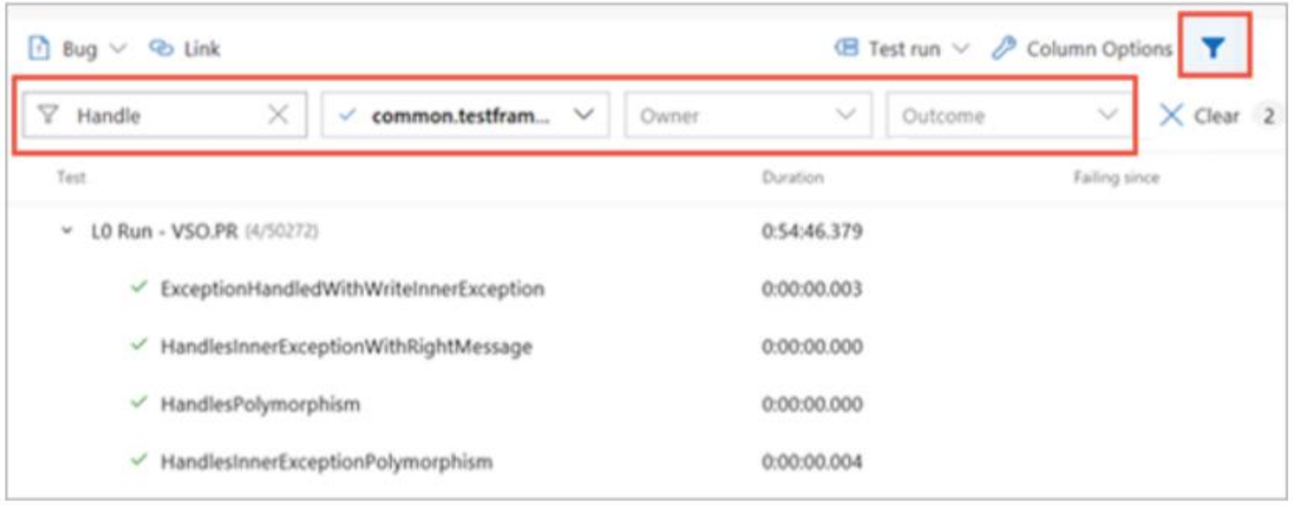
Test	Duration	Failing since
✓ TestRun_PartsUnlimited_20190814.2 (16/16)	0:00:07.357	
✓ AzureMLRecommendation_Exception	0:00:00.533	
✓ AzureMLRecommendation_Result	0:00:00.297	
✓ Home_Index	0:00:00.213	

Surface Test Results in the Tests Tab

- Tests results can be surfaced in the **Tests** tab using one of the following options:
  - **Test execution tasks**: built-in test execution tasks such as Visual Studio Test that automatically publish test results to the pipeline, or others such as Ant, Maven, Gulp, Grunt, and Xcode that provide this capability as an option within the task.
  - **Publish Test Results task**: publishes test results to Azure Pipelines when tests are executed using your choice of runner, and results are available in any of the supported test result formats.
  - **API(s)**: test results published directly using the Test Management API(s).

Tests Tab - Filter

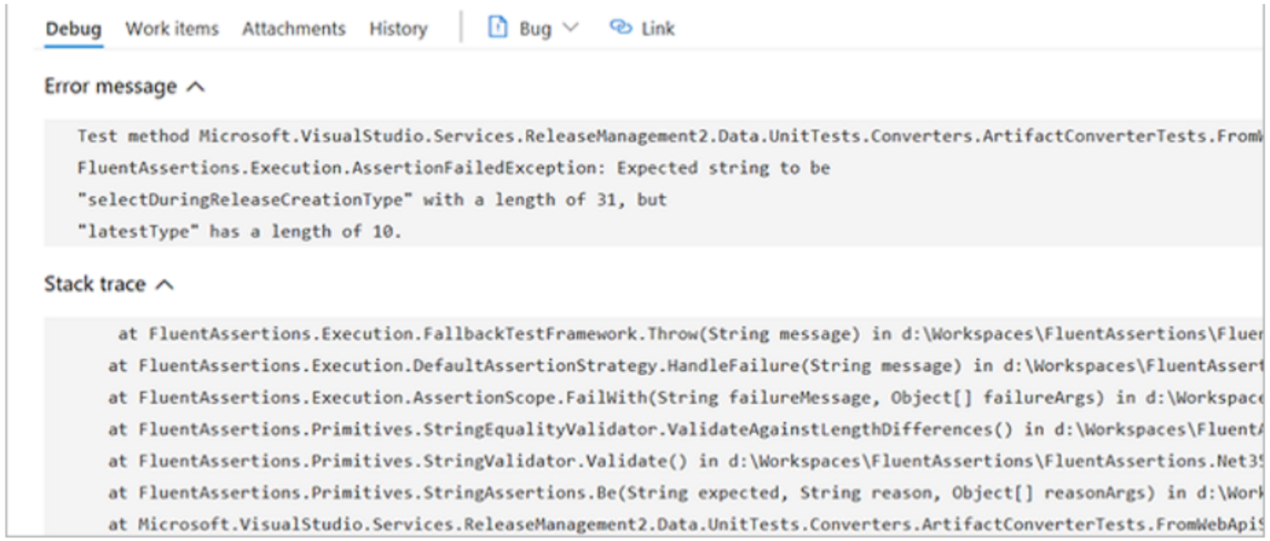
- Filters make it easy to quickly navigate to the test results of your interest.
- You can filter on **Test Name**, **Outcome** (failed, passed, and more), **Test Files** (files holding tests) and **Owner** (for test files).
- Additionally, with multiple **Grouping** options such as **Test run**, **Test file**, **Priority**, **Requirement**, and more, you can organize the **Results** view exactly as you require.



Test	Duration	Failing since
L0 Run - VSO.PR (4/50272)	0:54:46.379	
✓ ExceptionHandledWithWriteInnerException	0:00:00.003	
✓ HandlesInnerExceptionWithRightMessage	0:00:00.000	
✓ HandlesPolymorphism	0:00:00.000	
✓ HandlesInnerExceptionPolymorphism	0:00:00.004	

Tests Tab - Troubleshoot

- Error messages and stack traces are lengthy in nature and need enough real estate to view the details during troubleshooting.
- To provide an immersive troubleshooting experience, the **Details** view can be expanded to full page view while still being able to perform the required operations in context, such as bug creation or requirement association for the selected test result.



Error message

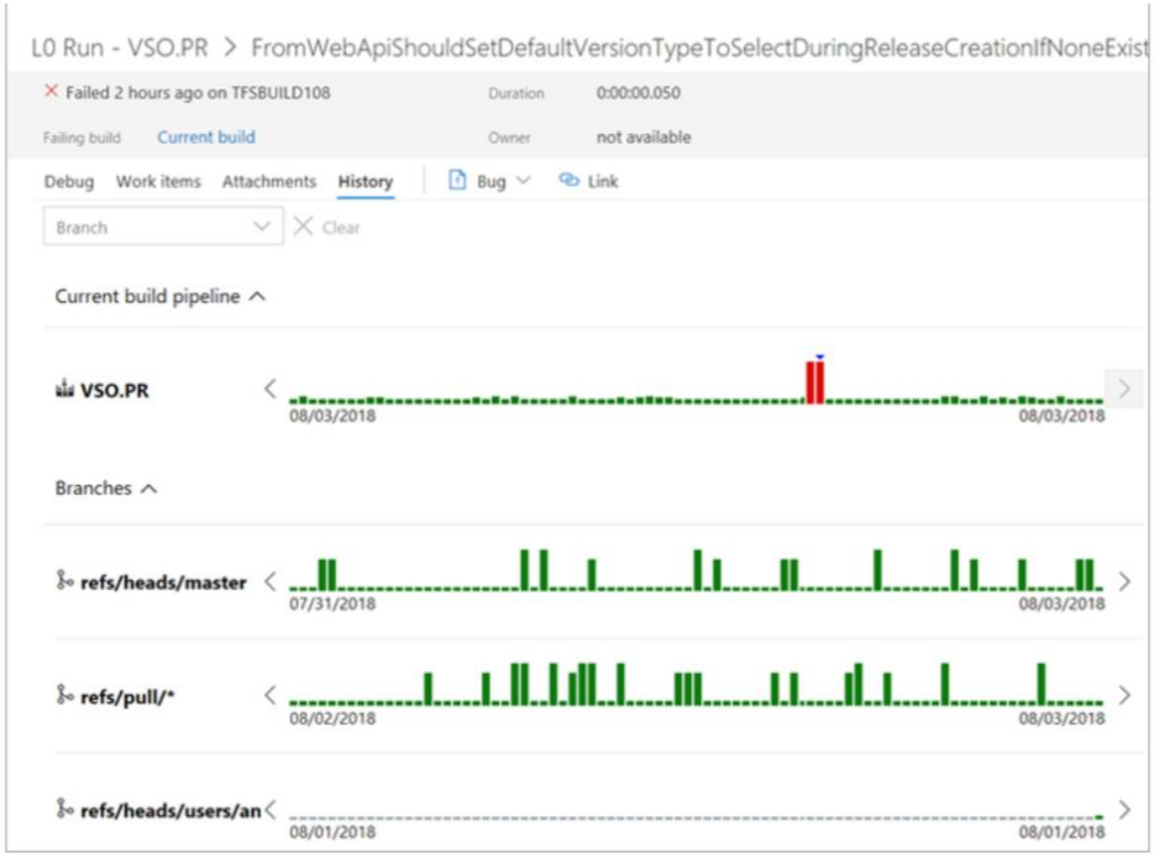
```
Test method Microsoft.VisualStudio.Services.ReleaseManagement2.Data.UnitTests.Converters.ArtifactConverterTests.From
FluentAssertions.Execution.AssertionFailedException: Expected string to be
"selectDuringReleaseCreationType" with a length of 31, but
"latestType" has a length of 10.
```

Stack trace

```
    at FluentAssertions.Execution.FallbackTestFramework.Throw(String message) in d:\Workspaces\FluentAssertions\Flue
   at FluentAssertions.Execution.DefaultAssertionStrategy.HandleFailure(String message) in d:\Workspaces\FluentAssert
   at FluentAssertions.Execution.AssertionScope.FailWith(String failureMessage, Object[] failureArgs) in d:\Workspace
   at FluentAssertions.Primitives.StringEqualityValidator.ValidateAgainstLengthDifferences() in d:\Workspaces\Fluent
   at FluentAssertions.Primitives.StringValidator.Validate() in d:\Workspaces\FluentAssertions\FluentAssertions.Net3
   at FluentAssertions.Primitives.StringAssertions.Be(String expected, String reason, Object[] reasonArgs) in d:\Work
   at Microsoft.VisualStudio.Services.ReleaseManagement2.Data.UnitTests.Converters.ArtifactConverterTests.FromWebApi
```

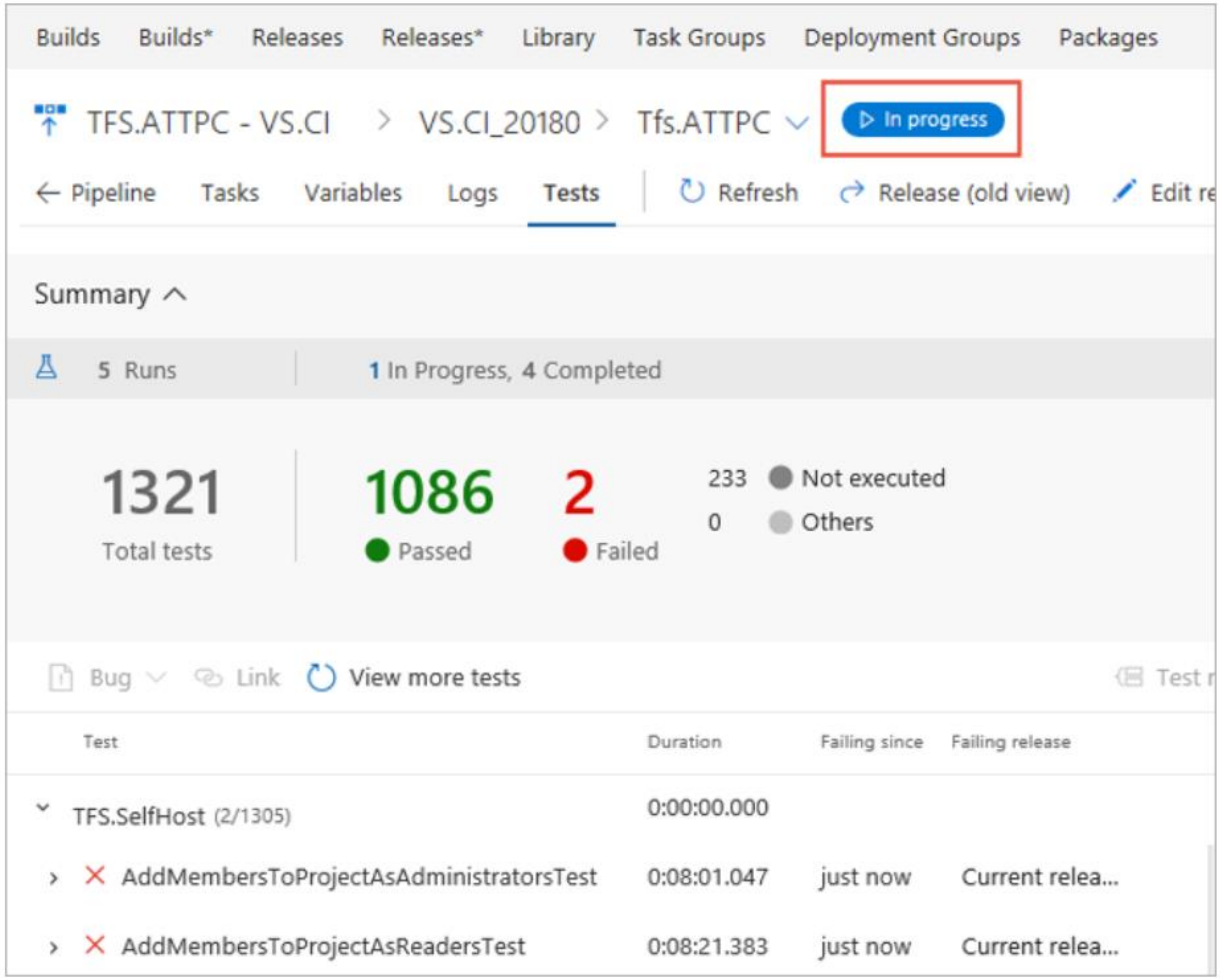
Tests Tab – Test Trends

- History of test execution can provide meaningful insights into reliability or performance of tests.
- When troubleshooting a failure, it is valuable to know how a test has performed in the past.
- The **Tests** tab provides test history in context with the test results.
- The test history information is exposed in a progressive manner starting with the current build pipeline to other branches, or the current stage to other stages, for build and release respectively.



Tests Tab – In Progress Tests

- Tests, such as integration and functional tests, can run for a long time. Therefore, it is important to see the current or near real-time status of test execution at any given time.
- The **In progress** view eliminates the need to wait for test execution to finish.
- Results are available in near real-time as execution progresses, helping you to take actions faster. You can debug a failure, file a bug, or abort the pipeline.



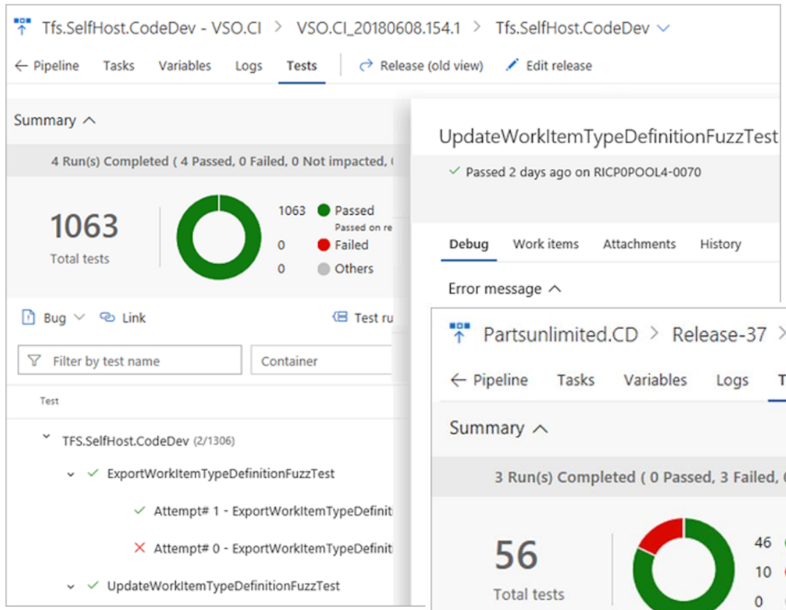
5 Runs | 1 In Progress, 4 Completed

1321 Total tests | 1086 ● Passed | 2 ● Failed | 233 ● Not executed | 0 ● Others

Test	Duration	Failing since	Failing release
TFS.SelfHost (2/1305)	0:00:00.000		
AddMembersToProjectAsAdministratorsTest	0:08:01.047	just now	Current relea...
AddMembersToProjectAsReadersTest	0:08:21.383	just now	Current relea...

Tests Tab – Summarized Results

- During test execution, a test might spawn multiple instances or tests that contribute to the overall outcome.
- Some examples are:
  - Tests that are rerun
  - Tests composed of an ordered combination of other tests (ordered tests)
  - Tests having different instances based on an input parameter (data driven tests).
- As these tests are related, they must be reported together with the overall outcome derived from the individual instances or tests.
- These test results are reported as a summarized test result in the **Tests** tab:
  - **Rerun failed tests**
  - **Data driven tests**


Tfs.SelfHost.CodeDev - VSO.CI > VSO.CI_20180608.154.1 > Tfs.SelfHost.CodeDev

← Pipeline | Tasks | Variables | Logs | **Tests** | Release (old view) | Edit release

Summary

4 Run(s) Completed (4 Passed, 0 Failed, 0 Not impacted,

1063 Total tests

1063 ● Passed
Passed on re
0 ● Failed
0 ● Others

Bug | Link | Test ru

Filter by test name | Container

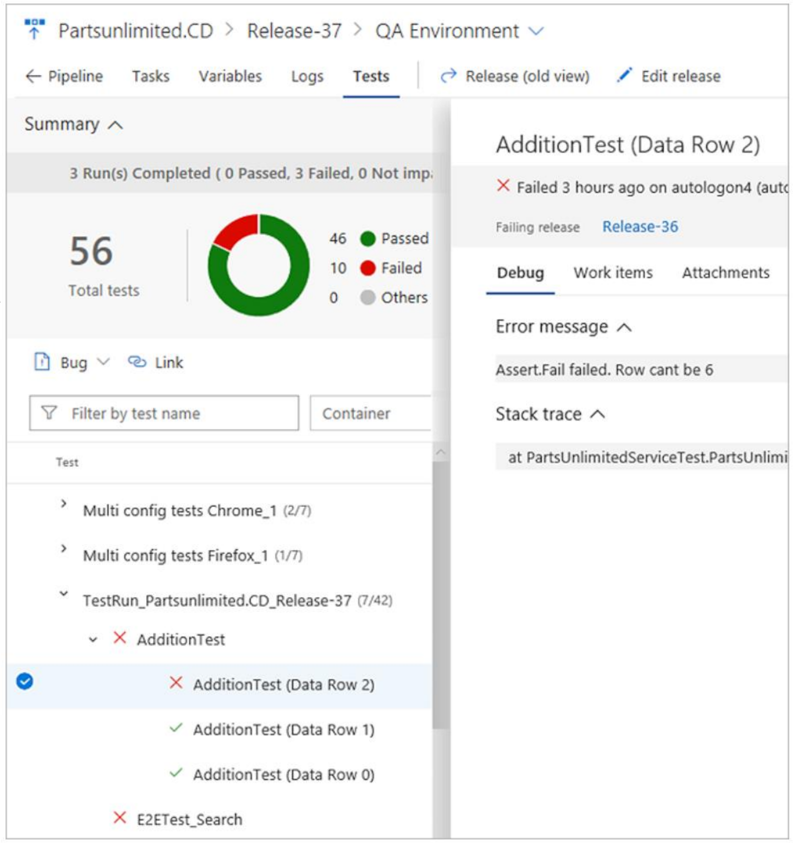
Test
- TFS.SelfHost.CodeDev (2/1306)
  - ✓ ExportWorkItemTypeDefinitionFuzzTest
    - ✓ Attempt# 1 - ExportWorkItemTypeDefinit
    - ✗ Attempt# 0 - ExportWorkItemTypeDefinit
  - ✓ UpdateWorkItemTypeDefinitionFuzzTest

UpdateWorkItemTypeDefinitionFuzzTest

✓ Passed 2 days ago on RICP0POOL4-0070

Debug | Work items | Attachments | History

Error message


Partsunlimited.CD > Release-37 > QA Environment

← Pipeline | Tasks | Variables | Logs | **Tests** | Release (old view) | Edit release

Summary

3 Run(s) Completed (0 Passed, 3 Failed, 0 Not imp

56 Total tests

46 ● Passed
10 ● Failed
0 ● Others

Bug | Link

Filter by test name | Container

Test
- Multi config tests Chrome_1 (2/7)
- Multi config tests Firefox_1 (1/7)
- TestRun_Partsunlimited.CD_Release-37 (7/42)
  - ✗ AdditionTest
    - ✗ AdditionTest (Data Row 2)
    - ✓ AdditionTest (Data Row 1)
    - ✓ AdditionTest (Data Row 0)
  - ✗ E2ETest_Search

AdditionTest (Data Row 2)

✗ Failed 3 hours ago on autologon4 (autc

Failing release Release-36

Debug | Work items | Attachments

Error message

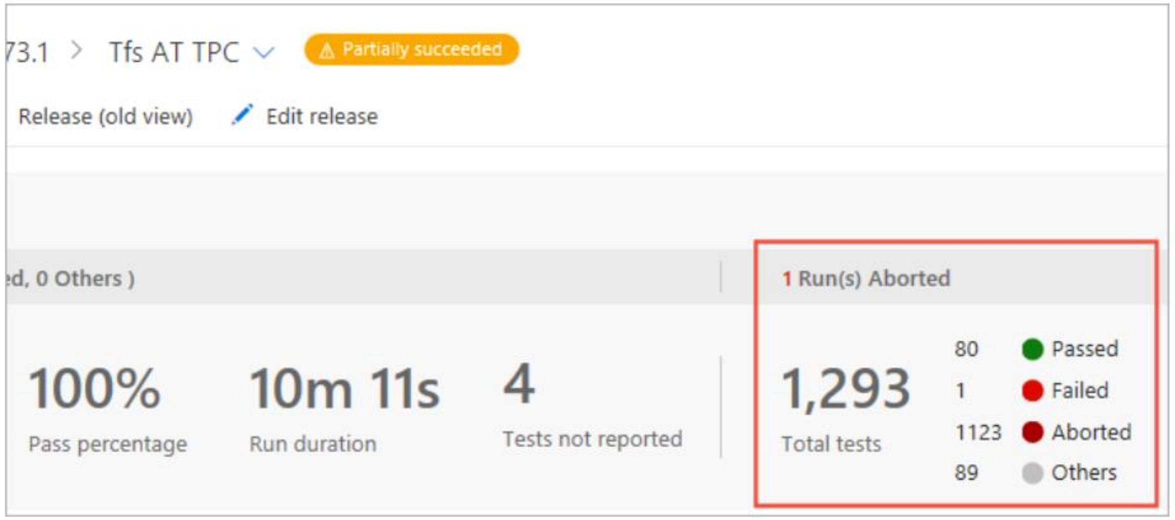
Assert.Fail failed. Row cant be 6

Stack trace

at PartsUnlimitedServiceTest.PartsUnlimi

Tests Tab – Aborted Tests

- Test execution can abort due to several reasons such as bad test code, errors in the source under test, or environmental issues.
- The aborted tests and test runs can be viewed alongside the completed runs in the **Tests** tab.


73.1 > Tfs AT TPC ⌄ ⚠ Partially succeeded

Release (old view) Edit release

d, 0 Others)

1 Run(s) Aborted

100% Pass percentage

10m 11s Run duration

4 Tests not reported

1,293 Total tests

80	●	Passed
1	●	Failed
1123	●	Aborted
89	●	Others

Surface Test Information Beyond the Tests Tab

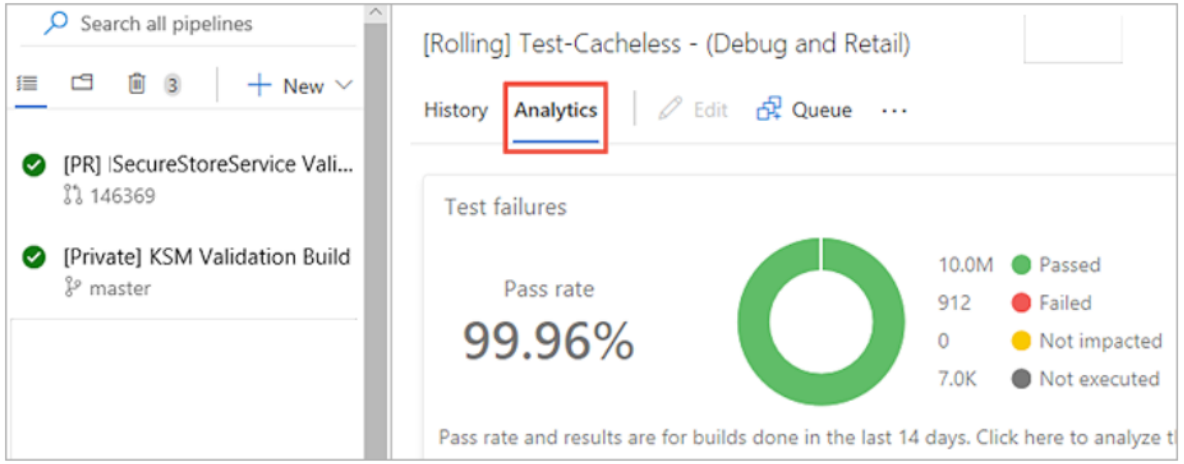
- The **Tests** tab provides a detailed summary of the test execution. This is helpful in tracking the quality of the pipeline, as well as troubleshooting failures.
- Azure DevOps Services also provides other ways to surface the test information:
    - The **Dashboard** provides visibility into your team's progress. Add one or more widgets that surface test related information:
        - Requirements quality
        - Test results trend
        - Deployment status
    - **Test analytics** provides rich insights into test results measured over a period of time. It can help identify problematic areas in your test by providing data such as the top failing tests, and more.

Analyze Test Results

- Tracking test quality over time and improving test collateral is key to maintaining a healthy DevOps pipeline.
- Test analytics provides near real-time visibility into your test data for builds and releases.
- It helps improve the efficiency of your pipeline by identifying repetitive, high impact quality issues.
- In order to view test analytics, install the **Analytics Extension** from the Marketplace.

Test Analytics - Builds

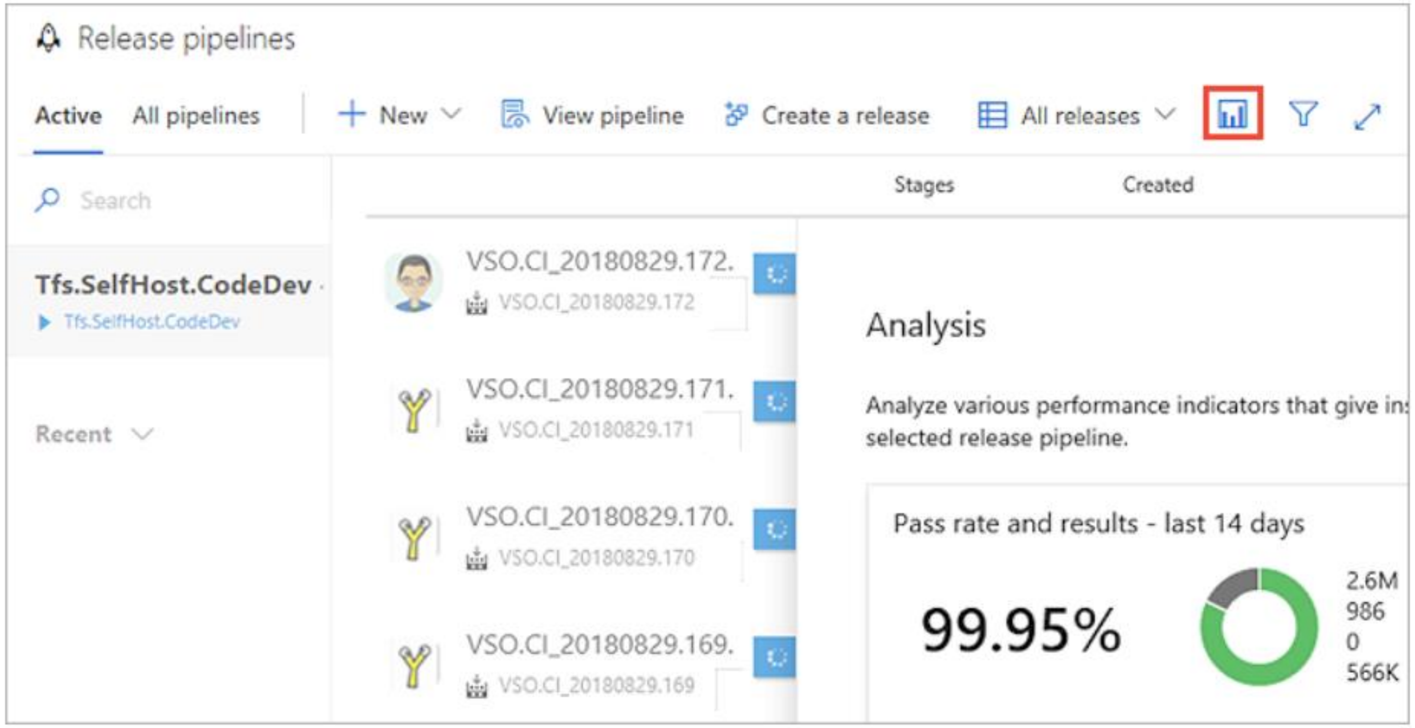
- To help teams find and fix tests that fail frequently or intermittently, use the **top failing tests** report.
- The build summary includes the **Analytics** page that hosts this report.
- The top-level view provides a summary of the test pass rate and results for the selected build pipeline, for the specified period.
- The default range is 14 days.



Pass rate and results are for builds done in the last 14 days. Click here to analyze t

Test Analytics - Releases

- For tests executing as part of release, access test analytics from the **Analytics** link at the top right corner.
- As with build, the summary provides an aggregated view of the test pass rate and results for the specified period.



Analysis

Analyze various performance indicators that give in selected release pipeline.

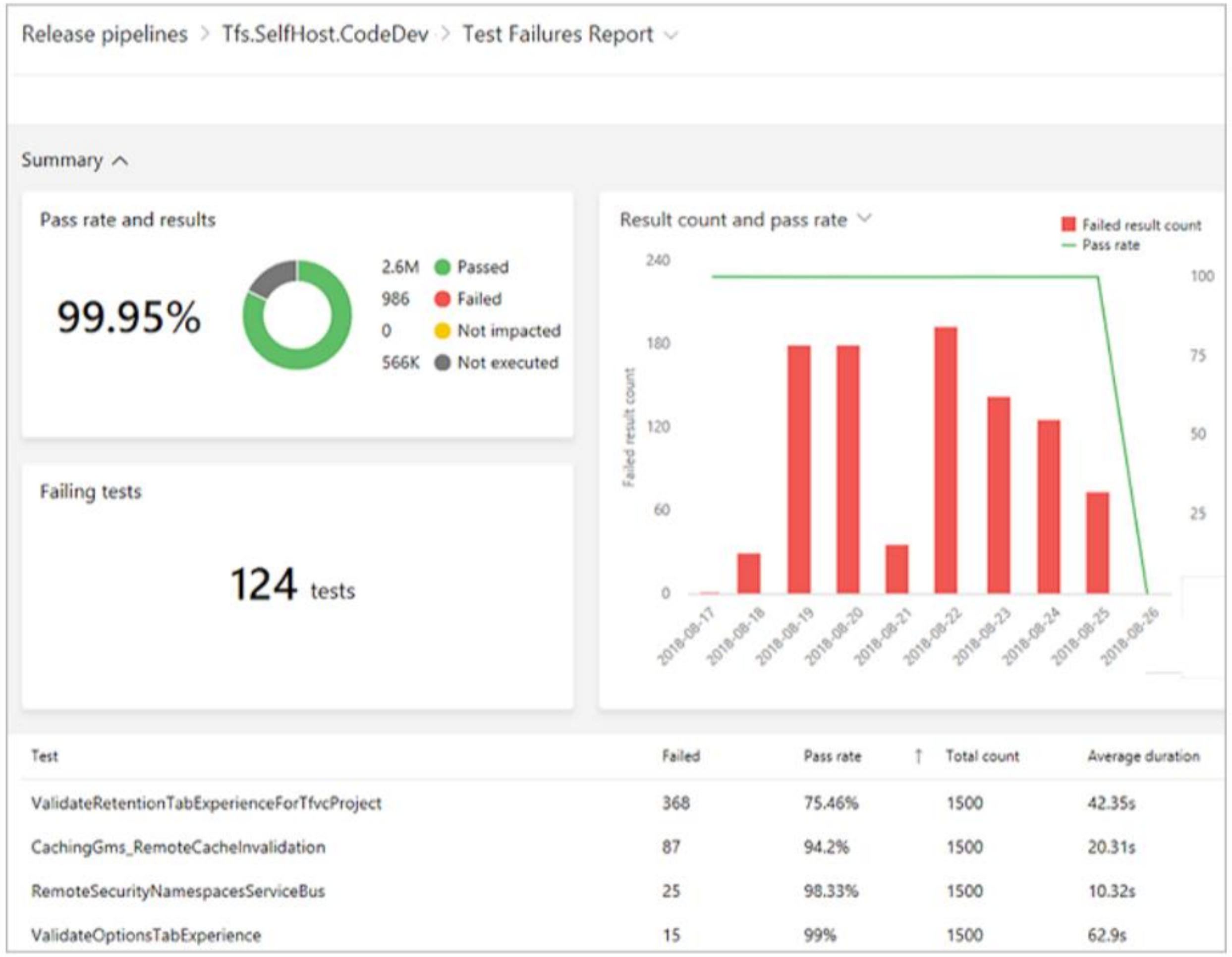
Pass rate and results - last 14 days

99.95%

2.6M
986
0
566K

Test Analytics – Test Failures

- Open a build or release summary to view the top failing tests report.
- This report provides a granular view of the top failing tests in the pipeline, along with the failure details.
- The report view can be organized in several different ways using the **group by** option.



Release pipelines > Tfs.SelfHost.CodeDev > Test Failures Report

Summary

Pass rate and results

99.95%

2.6M	🟢 Passed
986	🔴 Failed
0	🟡 Not impacted
566K	⚫ Not executed

Failing tests

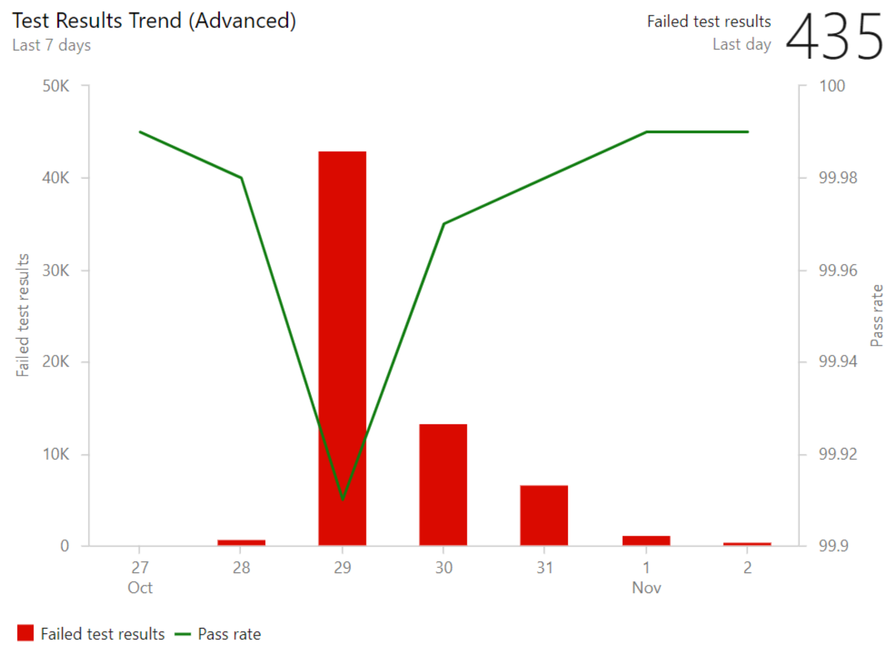
124 tests

Result count and pass rate

Test	Failed	Pass rate	Total count	Average duration
ValidateRetentionTabExperienceForTfvcProject	368	75.46%	1500	42.35s
CachingGms_RemoteCacheInvalidation	87	94.2%	1500	20.31s
RemoteSecurityNamespacesServiceBus	25	98.33%	1500	10.32s
ValidateOptionsTabExperience	15	99%	1500	62.9s

Test Result Trend (Advanced) Widget

- The **Test result trend (Advanced) widget** is now available for those who have installed the Analytics Extension on their Azure DevOps organization.
- It provides near real-time visibility into your test data for multiple builds and releases.
- The **Test result trend (Advanced) widget** displays a trend of your test results for your pipelines or across pipelines. You can use it to track the daily count of test, pass rate, and test duration. Tracking test quality over time and improving test collateral is key to maintaining a healthy DevOps pipeline.



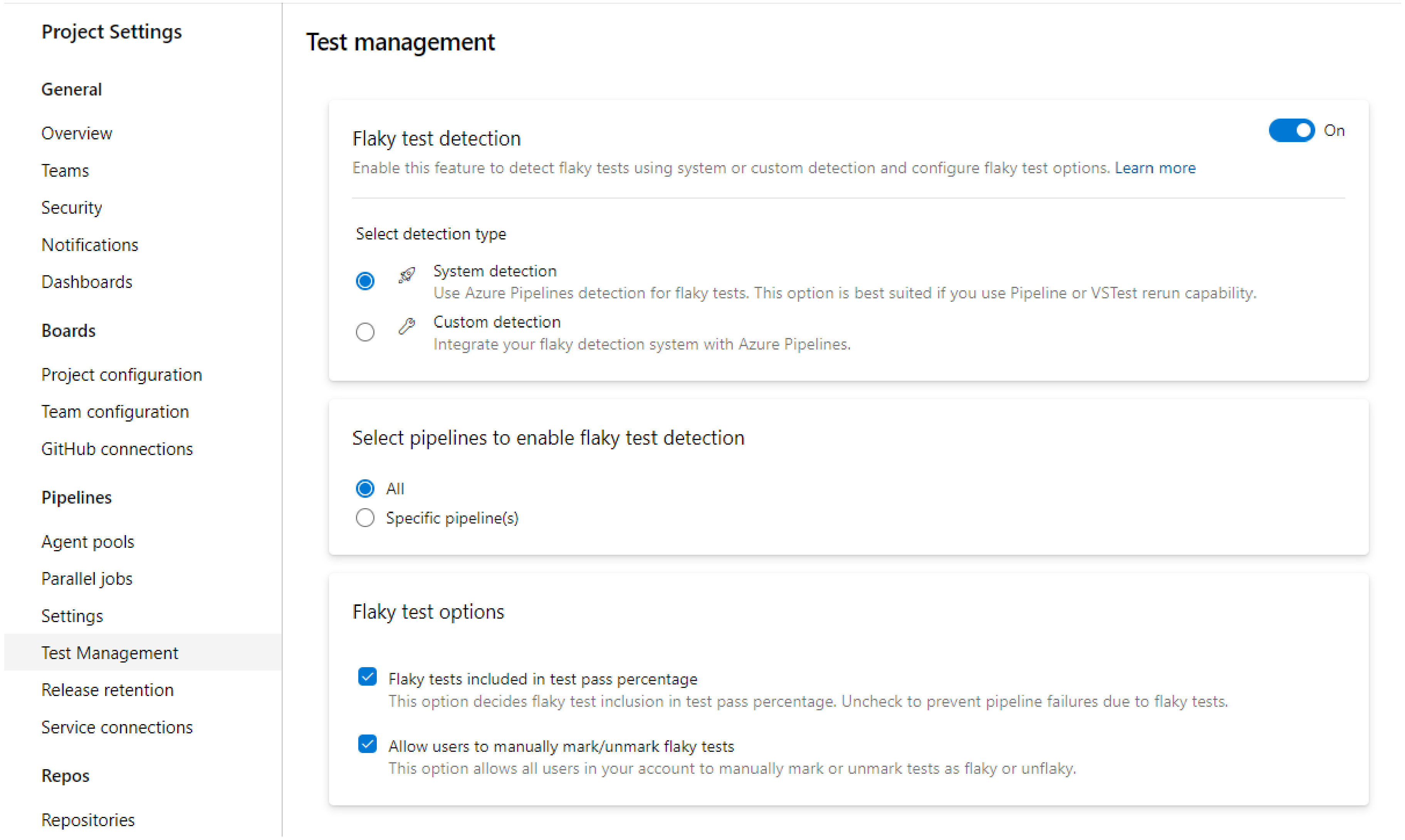
Trace Test Requirements

- **Requirements traceability** is the ability to relate and document two or more phases of a development process, which can then be traced both forward or backward from its origin.
- Requirements traceability help teams to get insights into indicators such as **quality of requirements** or **readiness to ship the requirement**.
- A fundamental aspect of requirements traceability is association of the requirements to test cases, bugs and code changes.
- Traceability standpoints:
    - **Quality Traceability**
    - **Bug Traceability**
    - **Source Traceability**

Flaky Test Management

- A flaky test is a test that provides different outcomes, such as pass or fail, even when there are no changes in the source code or execution environment.
- Flaky tests can affect developers' productivity since test failures may not be related to the changes under test. They can also impact the quality of shipped code.
- Flaky test management supports end-to-end lifecycle with detection, reporting and resolution. It supports system and custom detection.
- System detection is available via VSTest task rerun capability. All further executions of test for the same branch are also marked flaky until its resolved and unmarked.
- By default, flaky test information is available as additional meta-data.

Flaky Test Management (continued)



Project Settings

General

Overview
Teams
Security
Notifications
Dashboards

Boards

Project configuration
Team configuration
GitHub connections

Pipelines

Agent pools
Parallel jobs
Settings
Test Management
Release retention
Service connections

Repos

Repositories

Test management

Flaky test detection — On

Enable this feature to detect flaky tests using system or custom detection and configure flaky test options. Learn more

Select detection type

[✓ System detection]
Use Azure Pipelines detection for flaky tests. This option is best suited if you use Pipeline or VSTest rerun capability.

Custom detection
Integrate your flaky detection system with Azure Pipelines.

Select pipelines to enable flaky test detection

[✓ All]
Specific pipeline(s)

Flaky test options

☑ Flaky tests included in test pass percentage
This option decides flaky test inclusion in test pass percentage. Uncheck to prevent pipeline failures due to flaky tests.

☑ Allow users to manually mark/unmark flaky tests
This option allows all users in your account to manually mark or unmark tests as flaky or unflaky.

Demo 3: Reviewing Test Results

Lesson Knowledge Check

1. What is a Flaky Test?
2. What extension is needed if you would like to review historical test data and track test quality over time?
3. What can you link to test results in order to obtain traceability?

Lesson Summary

- In this lesson, you learned about:
  - Glossary
  - Review Test Results
  - View Test Results in Build
  - View Test Results in Release
  - Tests Tab
  - Surface Test Results in the Tests Tab
  - Surface Test Information Beyond the Tests Tab
  - Test Analytics
  - Test Result Trend (Advanced) Widget
  - Trace Test Requirements
  - Flaky Test Management

Module Summary

- In this module, you learned about:
    - Continuous Testing using Azure Pipelines
    - Running Tests
    - Reviewing Test Results

Microsoft