



Code Coverage – Implementation

In this Lab, I will now introduce **code coverage** functionality in our pipeline.

Since we already have a unit test in place that verifies some functionality within our **main web application**, we can now collect **code coverage statistics** to analyze how much of our application code is actually being tested.

Modifying the Pipeline to Include Code Coverage

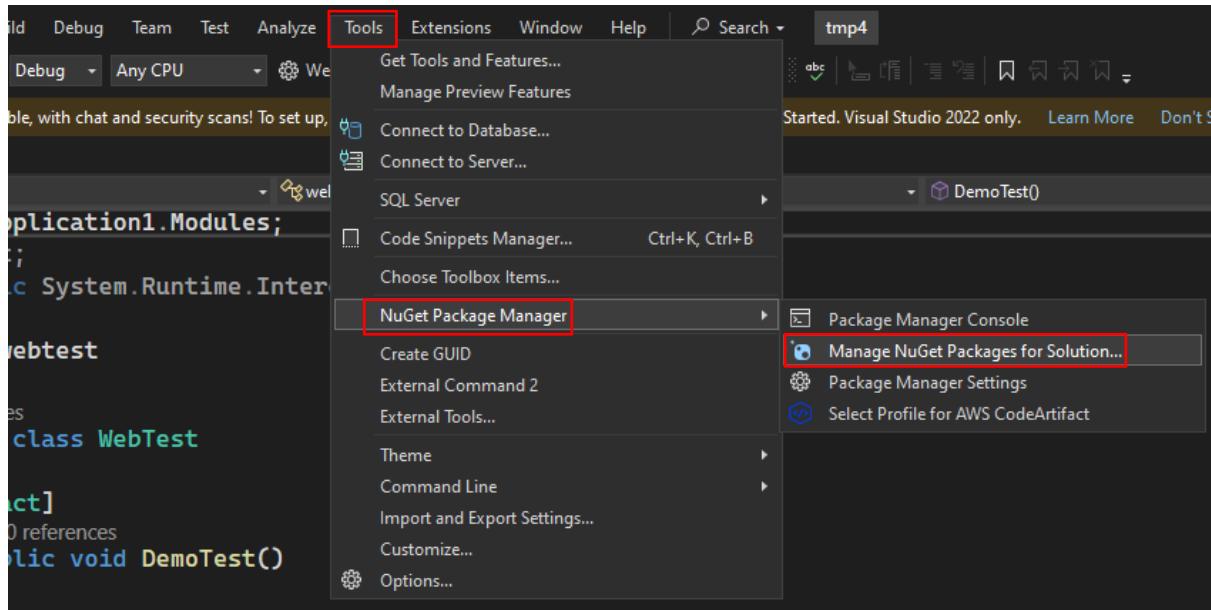
To enable code coverage, I need to make changes to my **Azure DevOps pipeline**:

1. Edit the Pipeline:

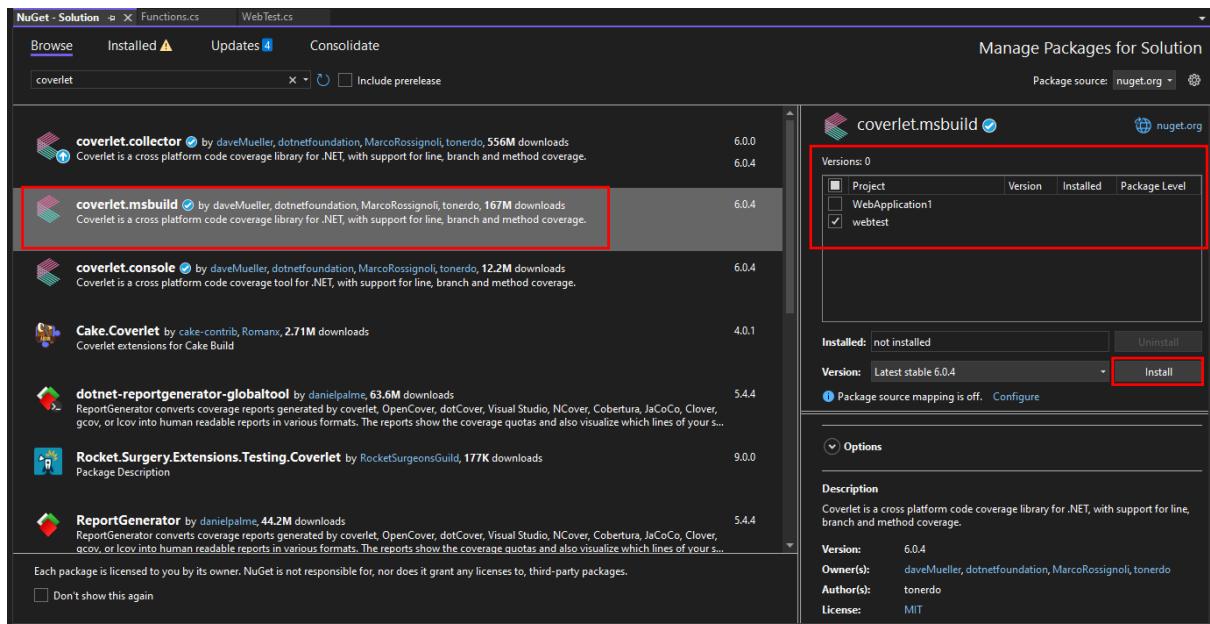
- Click on the **Edit Pipeline** option.
- Modify the pipeline by adding a new task **after the build stage** to restore dependencies.
- You can find all the **changes inside** the Files attached to this **Lab on GitHub**.

2. Why Restore Dependencies?

- As part of this change, I need to **install a NuGet package**.
- Open the Visual Studio for your project and click on Tools then choose NuGet Package Manager and click on Manager NuGet Packages for a solution.



- In the **Browse** section, search for `coverlet.msbuild` and install it.

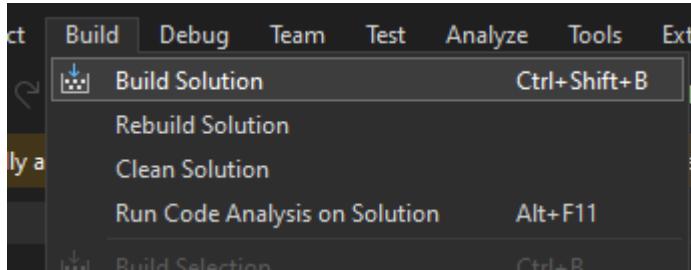


- This package helps in generating **code coverage reports**.

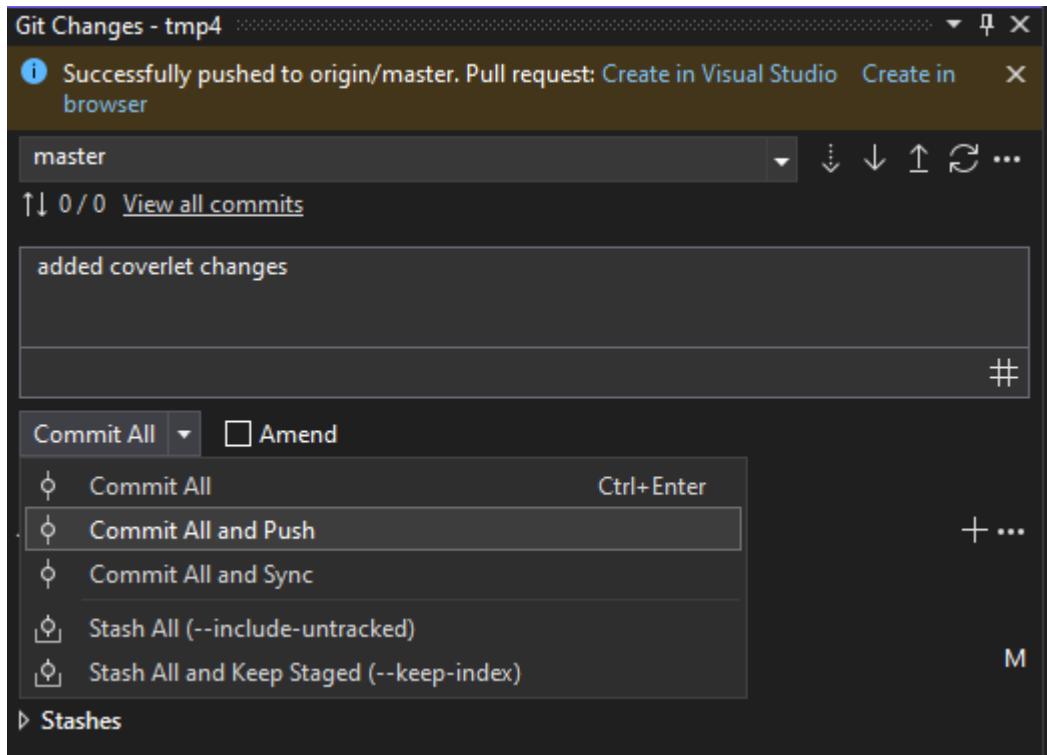
Building and Testing with Code Coverage Enabled

1. Rebuild the Solution:

- After installing the package, build the solution to apply the changes.



- Commit and push these changes to the **Git repository**.



2. Modify the Test Arguments:

- In the pipeline, update the **test task** to include additional arguments for collecting code coverage:
 - **Enable code coverage collection** by setting the option to true.
 - **Specify an output format** for the coverage results.
 - **Define an output folder** where the coverage data will be stored.

```

Settings
36 -> - task: DotNetCoreCLI@2
37   | inputs:
38   |   command: test
39   |   projects: '**/webtest.csproj'
40   |   arguments: '/p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./MyCoverage'
41   |   publishTestResults: true
42

```

Publishing Code Coverage Results

1. Add a New Task:

- Introduce a **task to publish code coverage results**.
- Set a **display name** for clarity.
- Specify the **code coverage tool format** (since we defined it earlier).
- Use the generated **XML file** to summarize the results.

2. Commit and Push Changes:

- Save all modifications and push them to the repository.

- This will trigger a new pipeline build automatically.
- To monitor progress, right-click the pipeline and open it in a new tab.

← WebApplication1.git



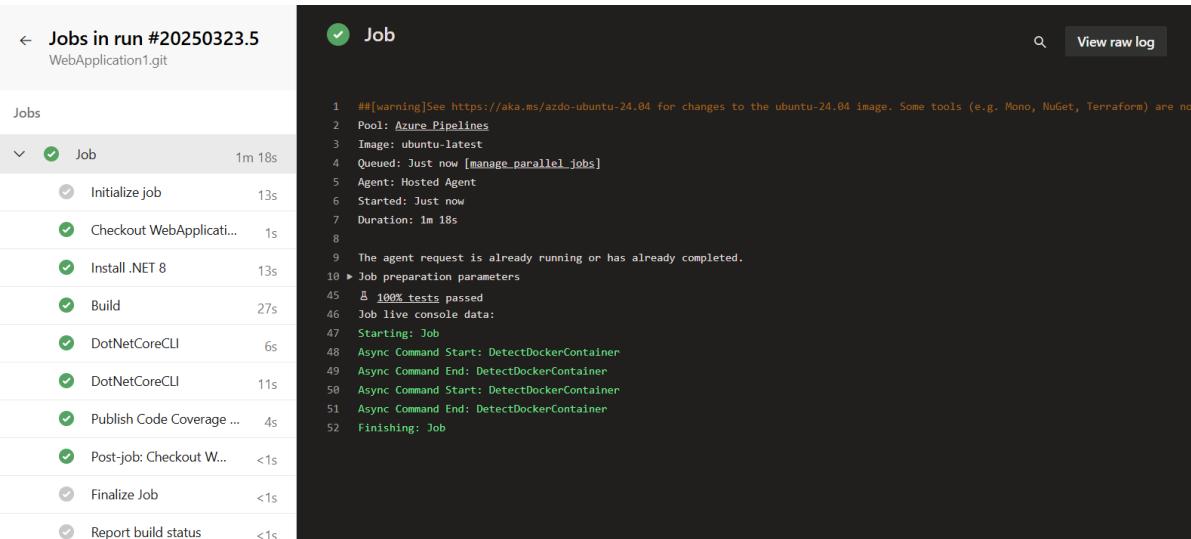
```

master ✘ WebApplication1.git / azure-pipelines.yml *

1 Settings
2 - task: DotNetCoreCLI@2
3   inputs:
4     command: 'restore'
5     projects: '**/*.csproj'
6 
7 Settings
8 - task: DotNetCoreCLI@2
9   inputs:
10    command: 'test'
11    projects: '**/webtest.csproj'
12    arguments: '/p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./MyCoverage/'
13    publishTestResults: true
14 
15 Settings
16 - task: PublishCodeCoverageResults@2
17   displayName: 'Publish Code Coverage Results'
18   inputs:
19     codeCoverageTool: 'Cobertura'
20     summaryFileLocation: '$(Build.SourcesDirectory)/**/MyCoverage/coverage.cobertura.xml'
21 
```

Viewing Code Coverage Results

- Once the pipeline run is complete, navigate to the test results.



The screenshot shows the Azure DevOps interface for a pipeline run. On the left, there's a list of jobs completed in the run, including 'Initialize job', 'Checkout WebApplication1.git', 'Install .NET 8', 'Build', 'DotNetCoreCLI', 'Publish Code Coverage ...', 'Post-job: Checkout W...', 'Finalize Job', and 'Report build status'. The total duration for the job was 1m 18s. On the right, the 'Job' tab is selected, displaying a log of the pipeline's execution. The log includes steps like 'Azure Pipelines' pool, 'ubuntu-latest' image, and various commands related to the build and test process. A note in the log says: 'The agent request is already running or has already completed.' The 'Code Coverage' tab is also visible at the bottom of the interface.

- A new "Code Coverage" tab will now appear.

#20250323.5 • New Coverlet code coverage pipeline
WebApplication1.git

This run is being retained as one of 3 recent runs by master (Branch).
View retention leases

Summary Tests **Code Coverage** Mend Bolt

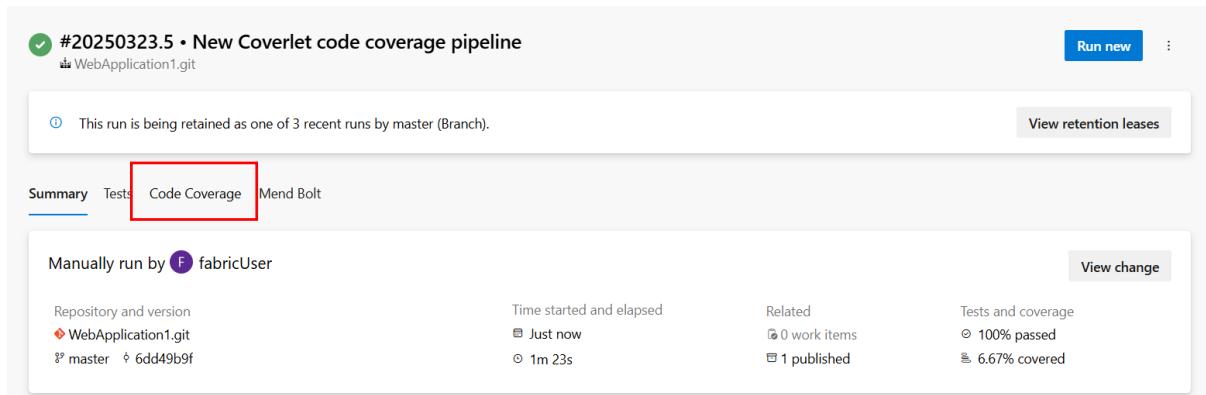
Manually run by fabricUser
View change

Repository and version
WebApplication1.git
master 6dd49b9f

Time started and elapsed
Just now
1m 23s

Related
0 work items
1 published

Tests and coverage
100% passed
6.67% covered



3. Inside the Code Coverage report:

- It displays the **total number of lines covered** by unit tests.
- Shows **line coverage percentage** for different parts of the application.
- Lists the **classes covered by unit tests**.

#20250323.5 • New Coverlet code coverage pipeline
WebApplication1.git

This run is being retained as one of 3 recent runs by master (Branch).
View retention leases

Summary Tests **Code Coverage** Mend Bolt

Summary

Sponsor Star

Information

Parser:	Cobertura
Assemblies:	1
Classes:	9
Files:	9
Coverage date:	03/23/2025 - 07:18:05

Line coverage

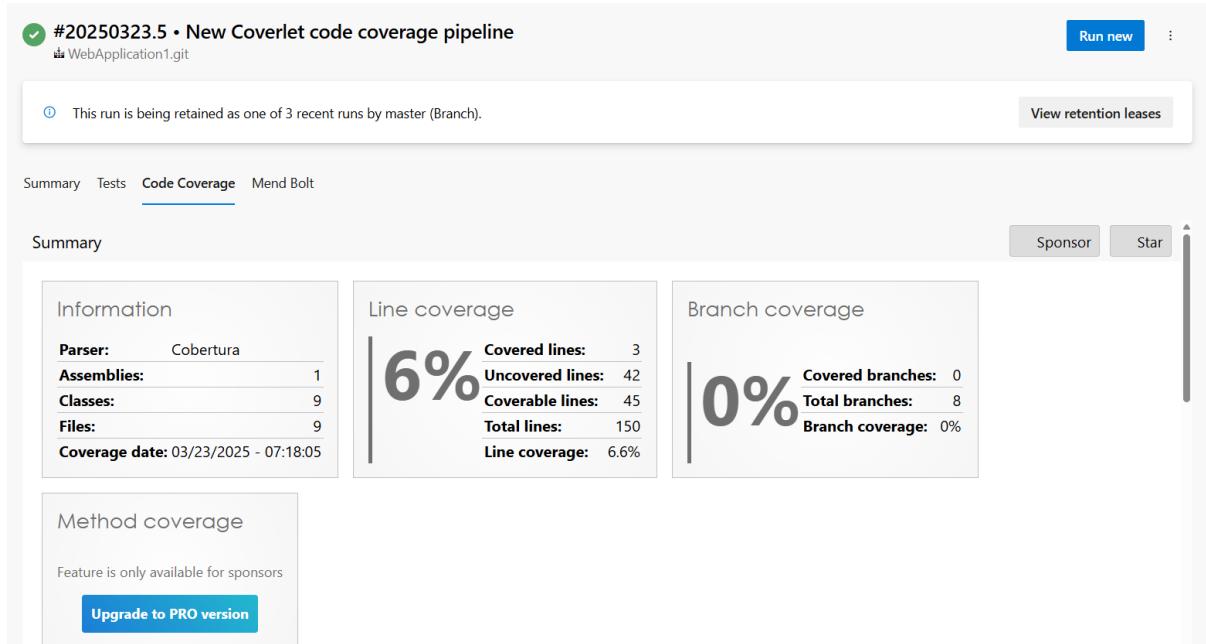
6%	Covered lines: 3
Uncovered lines: 42	
Coverable lines: 45	
Total lines: 150	
Line coverage: 6.6%	

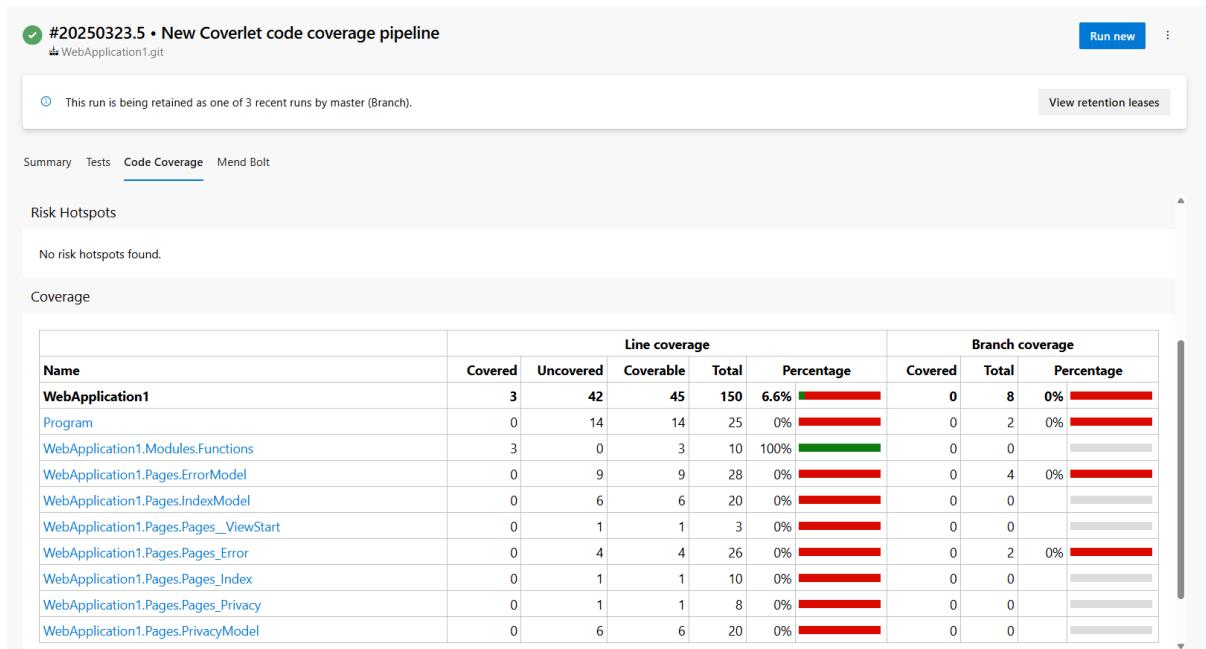
Branch coverage

0%	Covered branches: 0
Total branches: 8	
Branch coverage: 0%	

Method coverage

Feature is only available for sponsors
[Upgrade to PRO version](#)





4. Example:

- The Functions class has **100% code coverage** because we wrote a unit test that verifies the Add method.
- The report highlights which classes have been tested and which haven't.

Conclusion

The purpose of **code coverage** is to ensure that **unit tests are effectively testing the application code**. By integrating code coverage into the **Azure DevOps pipeline**, we can continuously track **test effectiveness** and identify areas that require additional testing.