😊 Code Coverage – Implementation

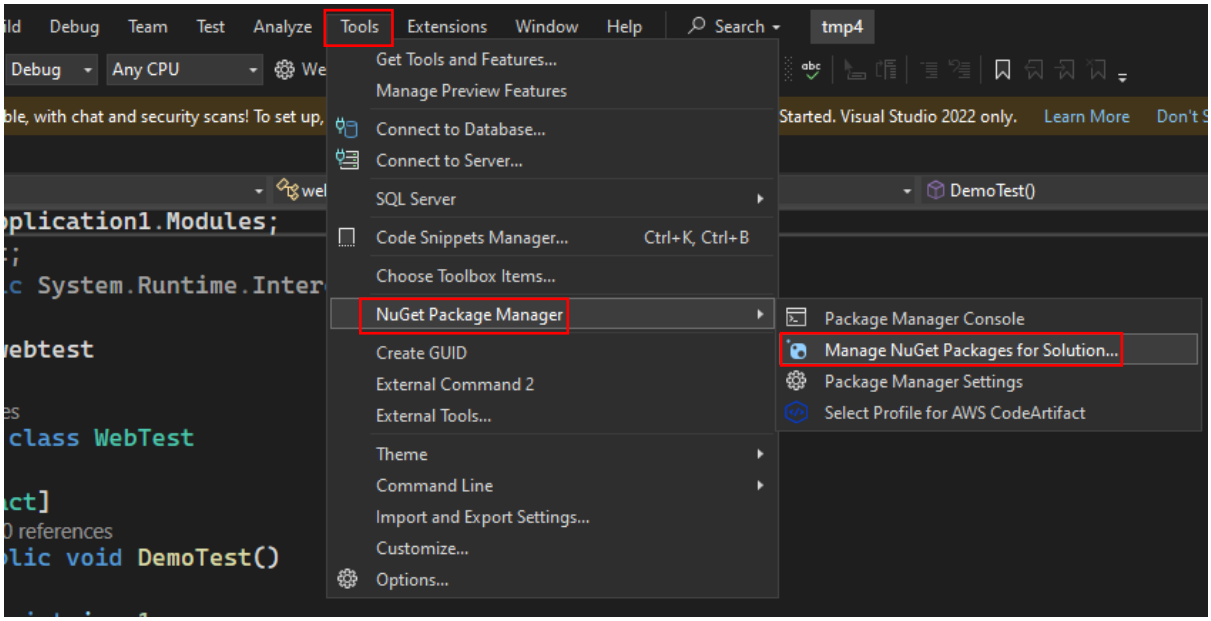
In this Lab, I will now introduce **code coverage** functionality in our pipeline.

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.

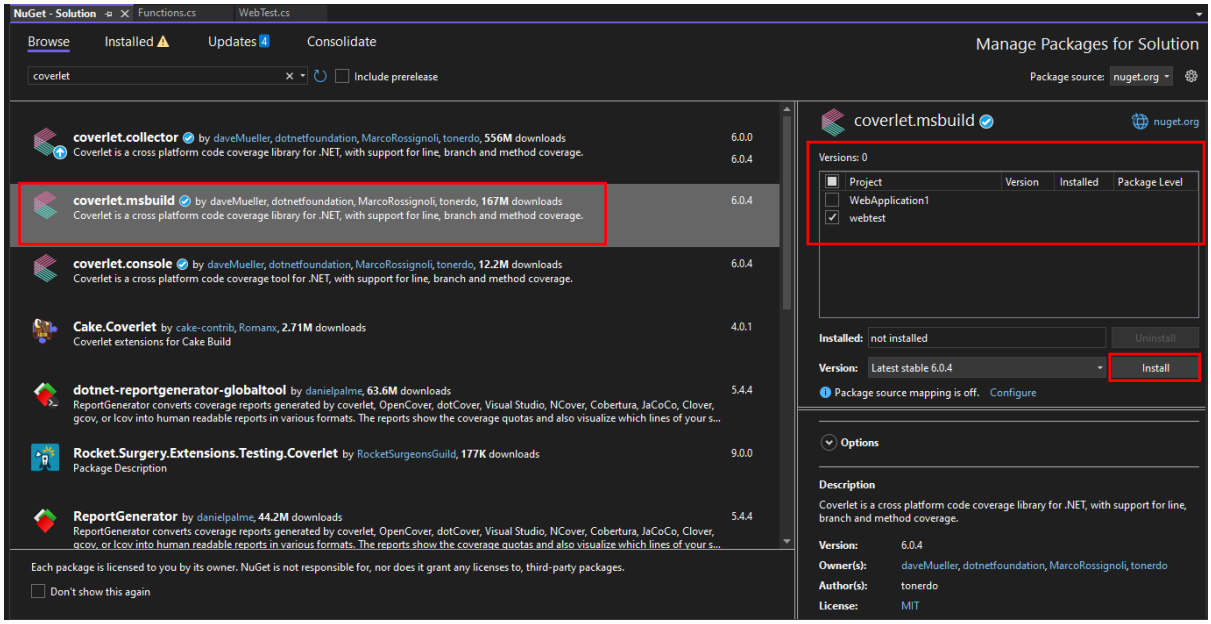
Modifying the Pipeline to Include Code Coverage

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.



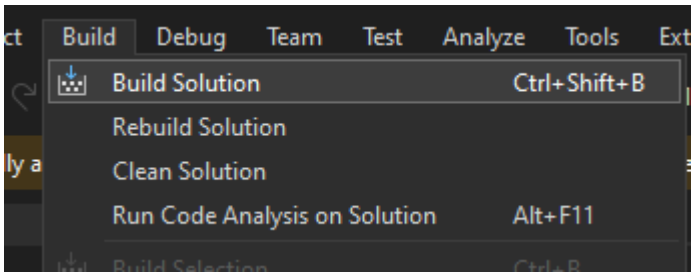
    - In the **Browse** section, search for coverlet.msbuild and install it.



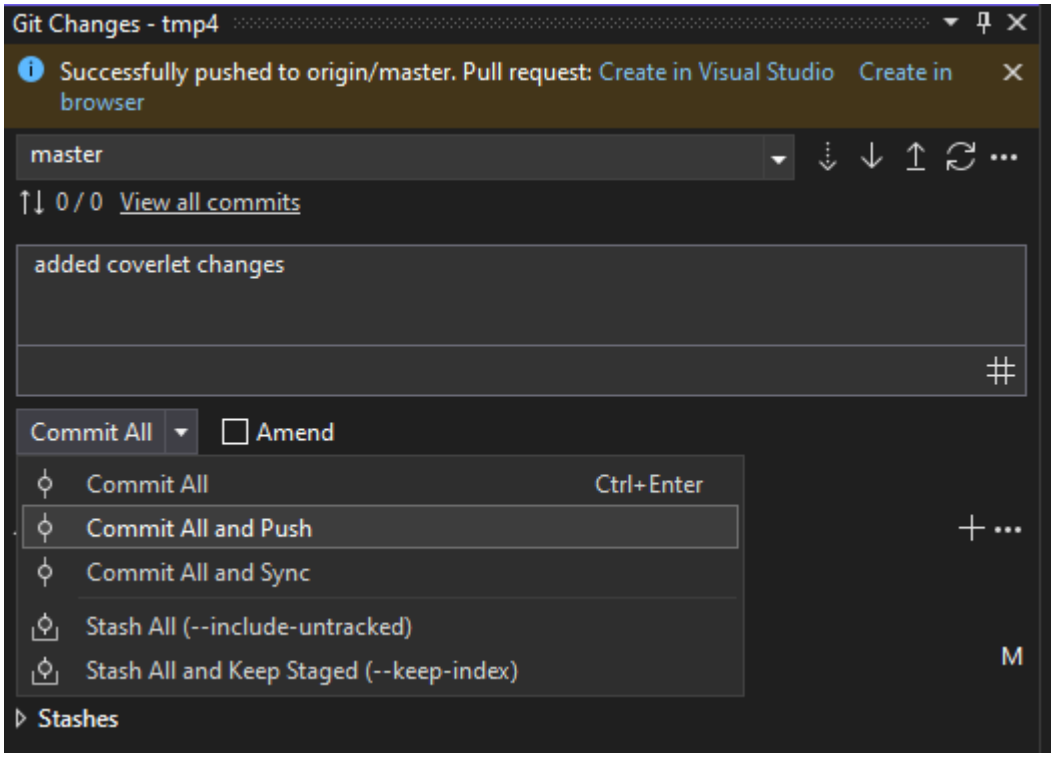
- This package helps in generating **code coverage reports**.

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.

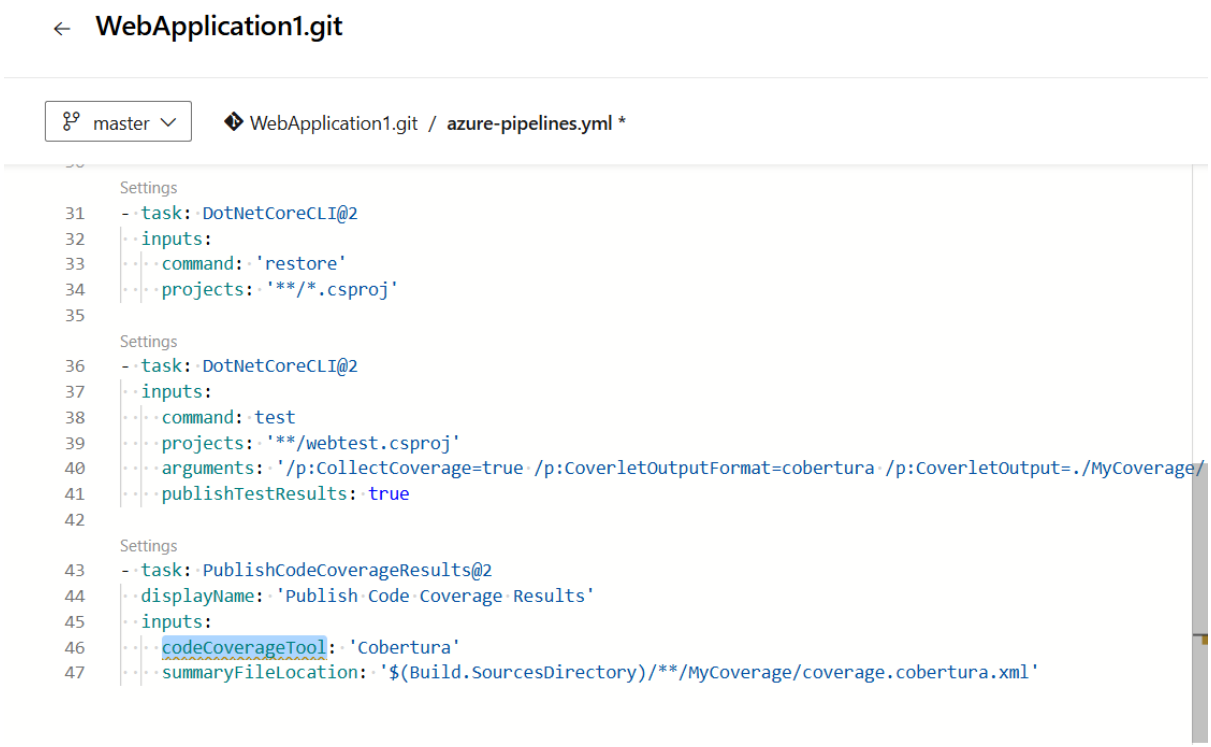
```
Settings
- task: DotNetCoreCLI@2
  inputs:
    command: test
    projects: '**/webtest.csproj'
    arguments: '/p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./MyCoverage/'
    publishTestResults: true

```

Publishing Code Coverage Results

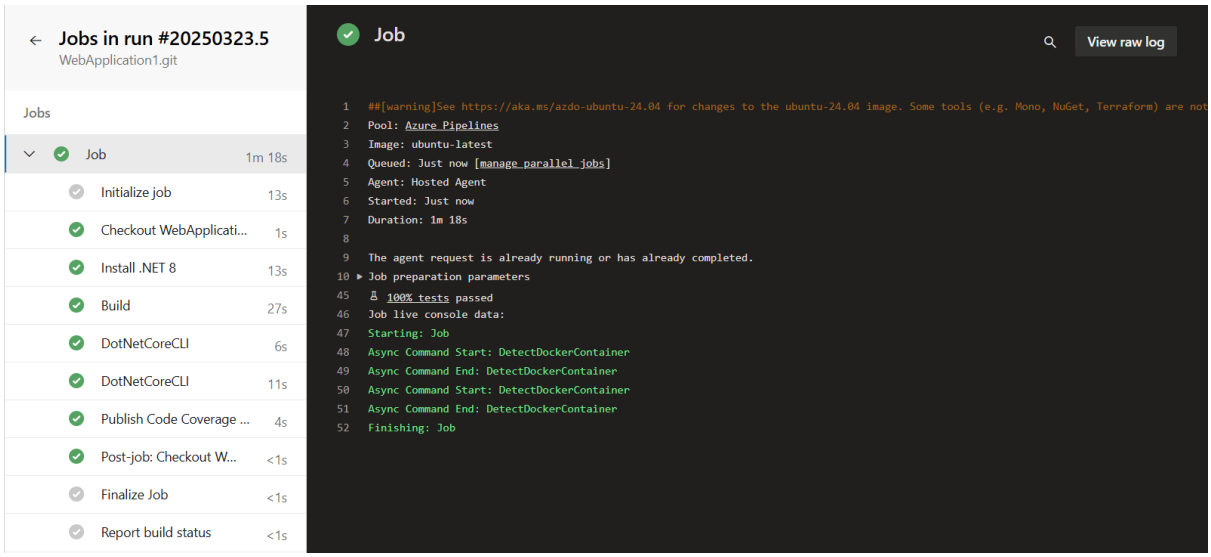
1. **Add a New Task:**
    - Introduce a **task to publish code coverage results**.
    - Set a **display name** for clarity.
    - Specify the **code coverage tool format** (since we defined it earlier).
    - Use the generated **XML file** to summarize the results.
2. **Commit and Push Changes:**
    - 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.

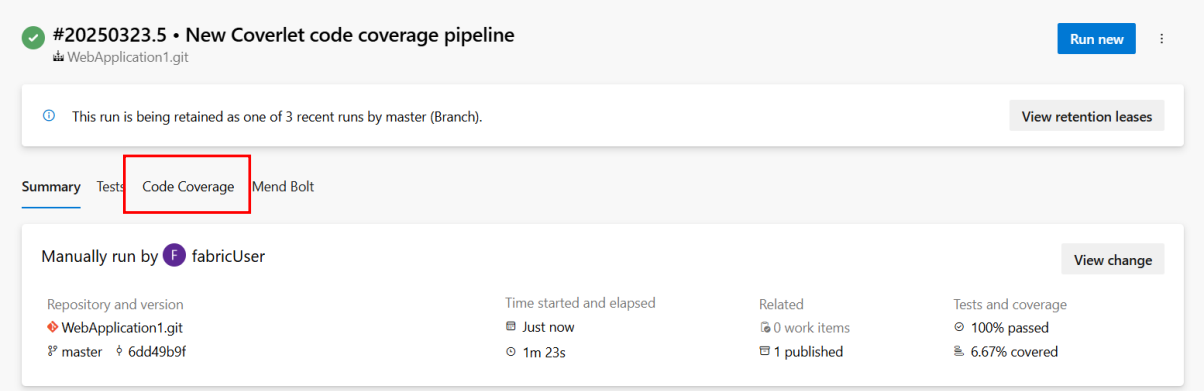


Viewing Code Coverage Results

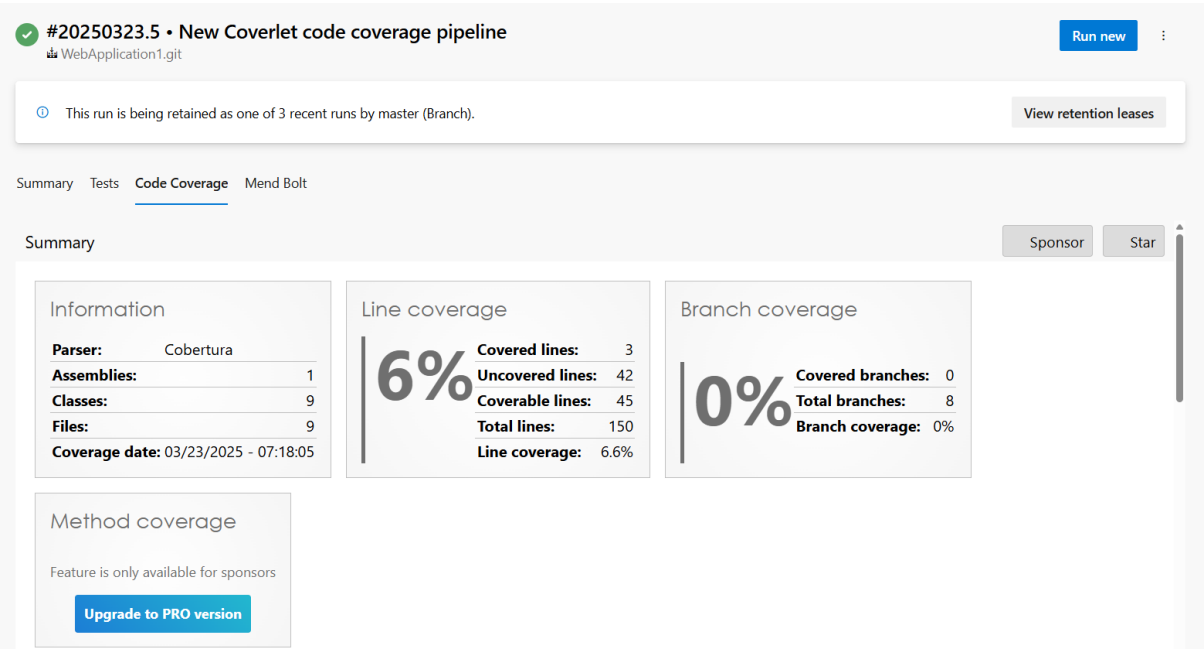
1. **Once the pipeline run is complete, navigate to the test results.**

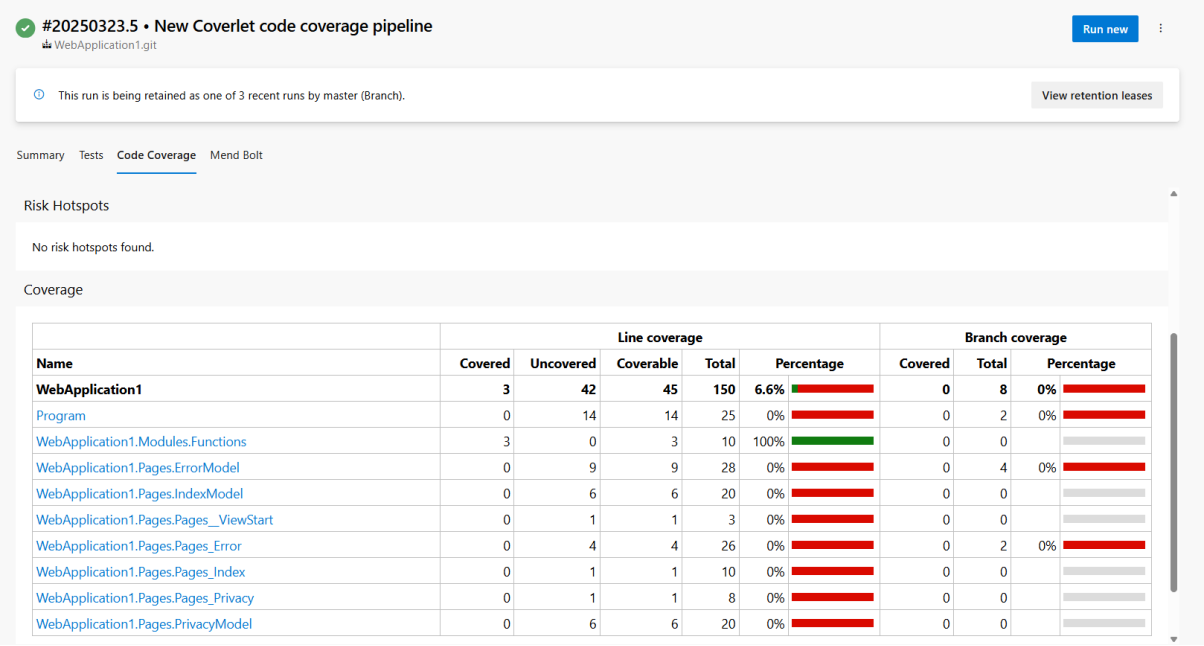


2. **A new "Code Coverage" tab will now appear.**



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**.





	Line coverage					Branch coverage		
Name	**Covered**	**Uncovered**	**Coverable**	**Total**	**Percentage**	**Covered**	**Total**	**Percentage**
WebApplication1	**3**	**42**	**45**	**150**	**6.6%**	**0**	**8**	**0%**
Program	0	14	14	25	0%	0	2	0%
WebApplication1.Modules.Functions	3	0	3	10	100%	0	0	
WebApplication1.Pages.ErrorModel	0	9	9	28	0%	0	4	0%
WebApplication1.Pages.IndexModel	0	6	6	20	0%	0	0	
WebApplication1.Pages.Pages__ViewStart	0	1	1	3	0%	0	0	
WebApplication1.Pages.Pages_Error	0	4	4	26	0%	0	2	0%
WebApplication1.Pages.Pages_Index	0	1	1	10	0%	0	0	
WebApplication1.Pages.Pages_Privacy	0	1	1	8	0%	0	0	
WebApplication1.Pages.PrivacyModel	0	6	6	20	0%	0	0	

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.