Lab requirements

- This lab requires **Microsoft Edge** or an Azure DevOps supported browser.
- **Set up an Azure DevOps organization:** If you don't already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at Create an organization or project collection.

Lab overview

In the context of Azure DevOps, the term **technical debt** represents suboptimal means of reaching tactical goals, which negatively affects the ability to achieve strategic objectives in software development and deployment. Technical debt affects productivity by making code hard to understand, prone to failures, time-consuming to change, and difficult to validate. Without proper oversight and management, technical debt can accumulate over time and significantly impact the overall quality of the software and the productivity of development teams in the longer term.

SonarCloud is a cloud-based code quality and security service. The main features of SonarCloud include:

- Support for 23 programming and scripting languages, including Java, JS, C#, C/C++, Objective-C, TypeScript, Python, ABAP, PLSQL, and T-SQL.
- There are thousands of rules to track down hard-to-find bugs and quality issues based on powerful static code analyzers.
- Cloud-based integrations with popular CI services, including Travis, Azure DevOps, BitBucket, and AppVeyor.
- Deep code analysis for exploring all source files in branches and pull requests, helping reach a green Quality Gate and promote the build.
- Speed and scalability.

In this lab, you'll learn how to integrate Azure DevOps with SonarCloud.

Note: Before you run this lab, ensure that you have the ability to run Azure Pipelines. Due to the change to public projects that took place in February 2021, access to pipelines will need to be requested: https://devblogs.microsoft.com/devops/change-in-azure-pipelines-grant-for-public-projects/

Objectives

After you complete this lab, you will be able to:

- Set up an Azure DevOps project and CI build to integrate with SonarCloud.
- Analyze SonarCloud reports.
- Integrate static analysis into the Azure DevOps pull request process.

Estimated timing: 60 minutes

Instructions

Exercise 0: Configure the lab prerequisites

In this exercise, you will set up the prerequisites for the lab, which consist of a team project based on the Sonar Scanning Examples repository.

Task 1: Create the team project

In this task, you will create a new Azure DevOps project based on the Sonar Scanning Examples repository repository.

1. On your lab computer, start a web browser, navigate to the **Azure DevOps portal** and sign in to your Azure DevOps organization.
2. In the **Azure DevOps portal**, in the upper right corner, click **+ New project**.
3. On the **Create new project** pane, in the **Project name** textbox, type **SonarExamples**, in the **Visibility** section, click **Public**, and then click **Create**.
4. **Note**: Unless you intend to sign up for a paid plan with SonarCloud, make sure that you set your Azure DevOps project to be public. If you *do* intend to sign up for a paid plan, then you can create a private project.
5. On the **SonarExamples** pane, in the vertical menu bar at the far left of the Azure DevOps portal, click **Repos**, on the **SonarExamples is empty. Add some code!** pane, and, in the **Import a repository** section, click **Import**.
6. On the **Import a Git repository** pane, ensure that **Git** appears in the **Repository type** dropdown list, in the **Clone URL**, type **https://github.com/SonarSource/sonar-scanning-examples.git**, and click **Import**.
7. **Note**: The scanning examples repository contains sample projects for a number of build systems and languages including C# with MSBuild, and Maven and Gradle with Java.

Task 2: Generate an Azure DevOps personal access token

In this task, you will generate an Azure DevOps personal access token that will be used to authenticate from the Postman app you will install in the next task of this exercise.

8. On the lab computer, in the web browser window displaying the Azure DevOps portal, in the upper right corner of the Azure DevOps page, click the **User settings** icon, in the dropdown menu, click **Personal access tokens**, on the **Personal Access Tokens** pane, and click **+ New Token**.
9. On the **Create a new personal access token** pane, click the **Show all scopes** link and, specify the following settings and click **Create** (leave all others with their default values):

Setting	Value
Name	**Managing technical debt with SonarCloud and Azure DevOps lab**
Scopes	**Custom defined**

Scope	**Code**
Permissions	**Read & write**

10. On the **Success** pane, copy the value of the personal access token to Clipboard.
11. **Note**: Make sure you record the value of the token. You will not be able to retrieve it once you close this pane.
12. On the **Success** pane, click **Close**.

Task 3: Install and configure the SonarCloud Azure DevOps extension
In this task, you will install and configure the SonarCloud Azure DevOps extension in your Azure DevOps project.

13. On your lab computer, start a web browser, navigate to the SonarCloud extension page on the Visual Studio Marketplace, click **Get it free**, ensure that the name of your Azure DevOps organization appears in the **Select an Azure Devops organization** dropdown list, and click **Install**.
14. Once the installation completes, click **Proceed to organization**. This will redirect the browser to the Azure DevOps portal displaying your organization's home page.
15. **Note**: If you do not have the appropriate permissions to install an extension from the marketplace, a request will be sent to the account administrator to ask them to approve the installation.
16. **Note**: The SonarCloud extension contains build tasks, build templates and a custom dashboard widget.
17. In the web browser window, navigate to the **SonarCloud home page** https://sonarcloud.io/.
18. On the SonarCloud home page, click **Log in**.
19. On the **Log in or Sign up to SonarCloud**, click **With Azure DevOps**.
20. When prompted whether to **Let this app access your info?**, click **Yes**. If prompted, select **Consent of behalf of your organization** and **Accept**.
21. **Note**: In SonarCloud, you will create an organization and, within it, a new project. The organization and project you set up in SonarCloud will mirror the organization and project that you set up in Azure DevOps.
22. Click **Import an organization from Azure**.
23. On the **Create an organization** page, in the **Azure DevOps organization name** textbox, type the name of your Azure DevOps organization, in the **Personal Access Token** textbox, paste the value of the token you recorded in the previous exercise, and click **Continue**.
24. In the **Import organization details** section, in the **Key** textbox, type a string of characters that will designate your organization and click **Continue**.

25. **Note**: The key must be unique within the SonarCloud system. Make sure that the green checkmark appears to the right of the **Key** textbox. This indicates that the key satisfies the uniqueness prerequisite.
26. In the **Choose a plan** section, select the plan that you intend to use for this lab (**free** suggested) and click **Create Organization**.
27. **Note**: You have now created the SonarCloud organization that mirrors your Azure DevOps organization.
28. **Note**: Next, within the newly created organization, you will create a SonarCloud project that will mirror the Azure DevOps project **SonarExamples**.
29. On the **Analyze projects - Select repositories** page, in the list of Azure DevOps projects, select the checkbox next to the **SonarExamples / SonarExamples** entry and click **Set up**.
30. On the **Choose your Analysis Method** page, click **With Azure Pipeline** tile.
31. On the **Analyze with Azure Pipelines** page, in the **Install our extension** section, click **Continue**.
32. **Note**: You can skip extension creation if you have already installed it.
33. On the **Add a new Sonarcloud Service Endpoint**, follow the steps mentioned on your Azure DevOps project, give the name **SonarSC** to the service connection, **check** the box for granting access to all pipelines and click **Verify and save**. Back on Sonarcloud website, click on **continue**.
34. On the **Analyze with Azure Pipelines** page, in the **Configure Azure Pipelines** section, click **.NET**. This will display a sequence of steps required to **Prepare Analysis Configuration**, **Run Code Analysis**, and **Publish Quality Gate Result**. You will need these instructions for the pipeline definition.
35. **Note**: Review the listing of steps to accomplish each of these objectives. You will implement them in the subsequent tasks.
36. **Note**: Record the value of the token necessary to set up the SonarCloud Service Endpoint and the value of the **Project Key** and **Project Name**.

Exercise 1: Set up an Azure Pipeline that integrates with SonarCloud

In this exercise, you will set up an Azure Pipeline that integrates with SonarCloud.

Note: We will set up a new build pipeline that integrates with SonarCloud to analyze the **SonarExamples** code.

Task 1: Initiate creation of the project build pipeline

In this task, you will begin creating the build pipeline for our project.

37. Switch to the web browser window displaying the **SonarExamples** pane in the Azure DevOps portal. Go to **Project settings** and change **Visibility** to private and **Save**.

38. **Note**: This need to be done if you followed the steps in Mod 00 to setup a parallel job for private projects only and your organization currently has no jobs available for public project.

Task 2: Create a pipeline by using the YAML editor
In this task, you will create a pipeline by using the YAML editor.

Note: Before you continue with configuration of the YAML pipeline, you will first create a service connection for SonarCloud.

39. In the vertical menu bar at the far left of the Azure DevOps portal, click **Pipelines** and then click **Create Pipeline**.
40. On the **Where is your code?** pane, click **Azure Repos Git**.
41. On the **Select a repository** pane, click **SonarExamples**.
42. On the **Configure your pipeline** pane, click **.NET Desktop** YAML template.
43. **Note**: This will automatically display the YAML editor with the template YAML file open. In order to configure it correctly you will need to adjust it (or replace it) so that matches the following file:
44. CodeCopy
45.

```
trigger:
- master

pool:
  vmImage: 'windows-latest'

variables:
  buildConfiguration: 'Release'
  buildPlatform: 'any cpu'

steps:
- task: NuGetToolInstaller@0
  displayName: 'Use NuGet 4.4.1'
  inputs:
    versionSpec: 4.4.1

- task: NuGetCommand@2
  displayName: 'NuGet restore'
  inputs:
    restoreSolution: 'SomeConsoleApplication.sln'

- task: SonarCloudPrepare@1
  displayName: 'Prepare analysis configuration'
```

```
    inputs:
      SonarCloud: 'SC'
      organization: 'myorga'
      scannerMode: 'MSBuild'
      projectKey: 'dotnet-framework-on-azdo'
      projectName: 'Sample .NET Framework project with Azure
DevOps'


  - task: VSBuild@1
    displayName: 'Build solution **\*.sln'
    inputs:
      solution: 'SomeConsoleApplication.sln'
      platform: '$(BuildPlatform)'
      configuration: '$(BuildConfiguration)'

  - task: VSTest@2
    displayName: 'VsTest - testAssemblies'
    inputs:
      testAssemblyVer2: |
       **\$(BuildConfiguration)\*Test*.dll
       !**\obj\**
      codeCoverageEnabled: true
      platform: '$(BuildPlatform)'
      configuration: '$(BuildConfiguration)'

  - task: SonarCloudAnalyze@1
    displayName: 'Run SonarCloud analysis'

  - task: SonarCloudPublish@1
    displayName: 'Publish results on build summary'
```

46. **Note**: You can download the file **net-desktop-sonarcloud.yml** from the SonarSource GitHub repository.

47. **Note**: The YAML pipeline needs to be modified by following the remaining steps in this task.

48. In the **NuGetCommand@2** task, replace `restoreSolution: 'SomeConsoleApplication.sln'` with `restoreSolution: '**\SomeConsoleApplication.sln'` to account for the fact that our solution is not located in the root of the repo.

49. In the **SonarCloudPrepare@1** task, click **Settings** option to open visual helper, choose the created **sonarSC** service connection from the dropdown and replace the value of the fields as proposed on the **Sonarcloud website > Configure Azure Pipeline** section. Click **Add** to include the changes to pipeline.
50. In the **VSBuild@1** task, replace `solution: 'SomeConsoleApplication.sln'` with `solution: '**\SomeConsoleApplication.sln'` to account for the fact that our solution is not located in the root of the repo.
51. On the **Review your pipeline YAML** pane, click **Save and Run** and, on the **Save and run** pane, click **Save and run**.
52. **Note**: Skip the next task if you completed this task in YAML editor.
53. Go to Azure Pipelines > Pipelines and click in **Sonarexample** pipeline, wait for the pipeline to finish.

Task 3: Check pipeline results

In this task, you will check pipeline results.

54. On the build run pane (either YAML or Classic one created before) , review the content of the **Summary** tab and then click the **Extensions** tab header.
55. **Note**: If you left the **Publish Quality Gate Result** task enabled, the **Extension** tab includes the summary of the SonarCloud analysis report.
56. On the **Extensions** tab, click the **Detailed SonarCloud report**. This will automatically open a new browser tab displaying the report on your SonarCloud project page.
57. **Note**: Alternatively, you could browse to you SonarCloud project.
58. Verify that the report does not include the Quality Gate results and note the reason for its absence.
59. **Note**: To be able to see the Quality gate result, after running he first report we need to set **New Code Definition**. This way, subsequent pipeline runs will include Quality Gate results.
60. On the **Overview** tab of the SonarCloud project (Sonarcloud website), click on **Administration** icon (left column) and **New Code**.
61. On the **New code** tab of the SonarCloud project, click **Previous version**.
62. Switch to the web browser window displaying the **SonarExamples** project pane in the **Azure DevOps portal** with the most recent build run, click **Run new** and, on the **Run pipeline** pane, click **Run**.
63. On the build run pane, review the content of the **Summary** tab and then click the **Extensions** tab header.
64. On the **Extensions** tab, click the **Detailed SonarCloud report**. This will automatically open a new browser tab displaying the report on your SonarCloud project page.

65. Verify that the report and Azure DevOps **extension** tab now **includes the Quality Gate result**.

66. **Note**: We have now created a new organization on SonarCloud and configured an Azure DevOps build to perform analysis and push the results of the build to SonarCloud.

Exercise 2: Analyze SonarCloud reports

In this exercise, you will analyze SonarCloud reports.

Task 1: Analyze SonarCloud reports

In this task, you will analyze SonarCloud reports.

67. On the **Overview** tab of the SonarCloud project, we see a summary for the report about the **main** branch. If you click **Main branch** icon (left column), and choose **Overall Code**, you will see a more detailed report.

68. **Note**: The page has metrics such as **Code Smells**, **Coverage**, **Duplications**, and **Size** (lines of code). The following table briefly explains each of these terms.

Terms	Description
Bugs	An issue that represents an error in code. If this has not broken yet, it will, and probably at the worst possible moment. This needs to be fixed
Vulnerabilities	A security-related issue which represents a potential backdoor for attackers
Code Smells	A maintainability-related issue in the code. Leaving it as-is means that, at best, maintainers will have a harder time than they should when making subsequent changes. At worst, they'll be so confused by the state of the code that they'll introduce additional errors as they make changes
Coverage	An indication of the percentage of code that is being validated by tests such as unit tests. To guard effectively against bugs, these tests should exercise or cover a large portion of your code
Duplications	The duplications decoration shows which parts of the source code are duplicated
Size	Provides the count of lines of code within the project including the number of statements, functions, classes, files and directories

69. **Note**: The letter displayed next to the bug count designates the **Reliability Rating**. In particular, the letter **C** indicates that there is at least 1 major bug in this code. For more information on Reliability Rating, refer to SonarQube documentation. You will also find there more information on rule types and see severities.

70. Click the number designating the count of **Bugs**. This will automatically display the content of the **Issues** tab.

71. On the right side of the **Issues** tab, click the large rectangle representing the bug to display the corresponding code.

72. **Note**: Review the error details in line number 9 of **Program.cs** file, including the recommendation stating **Change this condition so that it does not always evaluate to 'true'; some subsequent code is never executed.**

73. Hover with the mouse pointer over vertical lines between the code and the line numbers to identify gaps in code coverage.

74. **Note**: Our sample project is very small and has no historical data. However, there are thousands of [public projects on SonarCloud] that have more interesting and realistic results.

Exercise 3: Implement Azure DevOps pull request integration with SonarCloud

In this exercise, you will set up pull request integration between Azure DevOps and SonarCloud.

Note: In order to configure SonarCloud analysis to perform analysis of code included in an Azure DevOps pull request, you need to perform the following tasks:

- Add an Azure DevOps personal access token to a SonarCloud project, which authorizes its access to pull requests.
- Configure an Azure DevOps branch policy that controls a pull request-triggered build

Task 1: Create an Azure DevOps personal access token for pull request integration with SonarCloud

In this task, you will review the personal access token requirements for implementing Azure DevOps pull request integration with a SonarCloud project.

75. Switch to the web browser window displaying the **SonarExamples** project in the Azure DevOps portal.
76. **Reuse** Azure DevOps personal access token you generated earlier in this lab or repeat the steps described earlier in this lab in order to generate a personal access token with the **Code** scope and **Read & write** permissions to the **SonarExamples** project.

77. **Note**: Alternatively, you can reuse the personal access token you generated earlier in this lab.

78. **Note**: SonarCloud comments to pull requests will be added in the security context of the user who created the personal access token. The recommended practice is to create a separate "bot" Azure DevOps user for this purpose, to clearly identify comments originating from SonarCloud.

Task 2: Configure pull request integration in SonarCloud

In this task, you will configure pull request integration in SonarCloud by assigning an Azure DevOps personal access token to your SonarCloud project.

79. Switch to the web browser window displaying the **SonarExamples** project in the **SonarCloud portal**.
80. On the project's dashboard page, click the icon for the **Administration** tab and, in the dropdown menu, click **General Settings**.
81. On the **General Settings** page, click **Pull Requests**.
82. In the **General** section of the **Pull Requests** settings, in the **Provider** dropdown list, select **Azure DevOps Services** and click **Save**.
83. In the **Integration with Azure DevOps Services** section of the **Pull Requests** settings, in the **Personal access token** textbox, paste the previously generated Azure DevOps personal access token and click **Save**

Task 3: Configure a branch policy for integration with SonarCloud
In this task, you will configure an Azure DevOps branch policy for integration with SonarCloud.

84. Switch to the web browser window displaying the **SonarExamples** project in the **Azure DevOps portal**.
85. In the vertical menu bar at the far left of the Azure DevOps portal, click **Repos** and, in the **Repos** section, click **Branches**.
86. On the **Branches** pane, in the list of branches, hover with the mouse pointer over the right edge of the **master** branch entry to reveal the vertical ellipsis character designating the **More options** menu, click it, and, in the popup menu, click **Branch policies**.
87. On the **master** pane, to the right of the **Build Validation** section, click **+**.
88. On the **Add build policy** pane, in the **Build pipeline** dropdown list, select the pipeline you created earlier in this lab, in the **Display name** textbox, type **SonarCloud analysis** and click **Save**.

89. **Note**: Azure DevOps is now configured to trigger a SonarCloud analysis when any pull request targeting the **master** branch is created.

Task 4: Validate pull request integration
In this task, you will validate pull request integration between Azure DevOps and SonarCloud by creating a pull request and reviewing the resulting outcome.

Note: You will make a change to a file in the repository and create a request to trigger SonarCloud analysis.

90. In the Azure DevOps portal, in the vertical menu bar on the left side, click **Repos**. This will display the **Files** pane.
91. In the central pane, in the folder hierarchy, navigate to the file **Program.cs** in the **sonarqube-scanner-msbuild\CSharpProject\SomeConsoleApplication** folder and click **Edit**.
92. On the **Program.cs** pane, add the following empty method to the code directly above the line `public static bool AlwaysReturnsTrue()`

```
C#Copy
    public void Unused(){

    }
```

95. On the **Program.cs** pane, click **Commit**.
96. On the **Commit** pane, in the **Branch name** textbox, type **branch1**, select the **Create a pull request** checkbox, and click **Commit**.
97. On the **New pull request** pane, select **Create**.
98. On the **Overview** tab of the **Updated Program.cs** pane, monitor the progress of the build process to its completion.
99. On the **Overview** tab of the **Updated Program.cs** pane, note that while required checks succeeded, there are optional checks that failed and click the **View 3 checks** link.
100. On the **Checks** pane, review the results of the SonarCloud checks and close the pane.
101. **Note**: The results show that the analysis builds completed successfully, but that the new code in the PR failed the Code Quality check. Comment has been posted to the PR regarding the new issue that was discovered.
102. Back on the **Overview** tab of the **Updated Program.cs** pane, scroll down to the section containing comments and review the comment from SonarCloud regarding the newly added class.
103. **Note**: The reported issues contain only changes in the code corresponding to the pull request. Pre-existing issues in **Program.cs** and other files are ignored.

Task 4: Block pull requests in response to failing Code Quality checks
In this task, you will configure blocking of pull requests in response to failing Code Quality checks.

Note: At this point, it is still possible to complete the pull request and commit the corresponding changes even though Code Quality checks fail. You will modify Azure DevOps configuration to block the commit unless the relevant Code Quality checks pass.

104. In the Azure DevOps portal, displaying the **Updated Programs.cs** pane, in the lower left corner, click **Project Settings**.
105. In the **Project Settings** vertical menu, in the **Repos** section, click **Repositories**.
106. On the **All repositories** pane, click **SonarExamples**.
107. On the **SonarExamples** pane, click the **Policies** tab header.
108. On the listing of **Policies** scroll down to the listing of branches and click the entry representing the **master** branch.
109. On the **master** pane, scroll down to the **Status Checks** section and click **+**.

110. On the **Add status policy** pane, in the **Status to check** dropdown list, select the **SonarCloud/quality gate** entry, ensure that the **Policy requirement** option is set to **Required**, and click **Save**

111. **Note**: At this point, users will not be able to merge pull request until the Code Quality check is successful. This, in turn, requires that all issues identified by SonarCloud have been either fixed or marked as **confirmed** or **resolved** in the corresponding SonarCloud project.

Review

In this lab, you learned how to integrate Azure DevOps Services with SonarCloud