Analyzing with SonarQube Extension for VSTS-TFS

By SonarSource – MIT – Issue Tracker – Sources

SonarQube/SonarCloud Extension for TFS-VSTS

Table of Contents

Features

The new structure of Team Foundation Build allows better integration with your build and release processes in Azure DevOps (Formerly VSTS and VSO) and Team Foundation Server (TFS) on-premise via a public extension which can be installed in your VSTS account or TFS server. The extension allows the analysis of all languages supported by SonarQube.

Compatibility

- Versions 3.x are compatible with:
  - TFS 2015 Update 3
  - TFS 2017 Update 1
  - VSTS
- Versions 4.x are compatible with:
  - TFS 2017 Update 2+
  - TFS 2018
  - Azure DevOps Server 2019
  - VSTS / Azure DevOps

The SonarQube Extension embeds its own version of the SonarQube Scanner for MSBuild.

Installation

1. Make sure the .NET Framework v4.6+ is installed
2. Make sure the Java Runtime Environment 8 is installed
3. Install the extension from the marketplace

TFS manual install

Note: if you are running on TFS earlier than 2017 Update 2, you will need to download and manually install the latest 3.x version of the VSIX.

- You can download the VSIX on the "Releases" page of the GitHub repository
- You can browse the documentation on SonarQube Extension 3.0

Use

Configure a SonarQube service

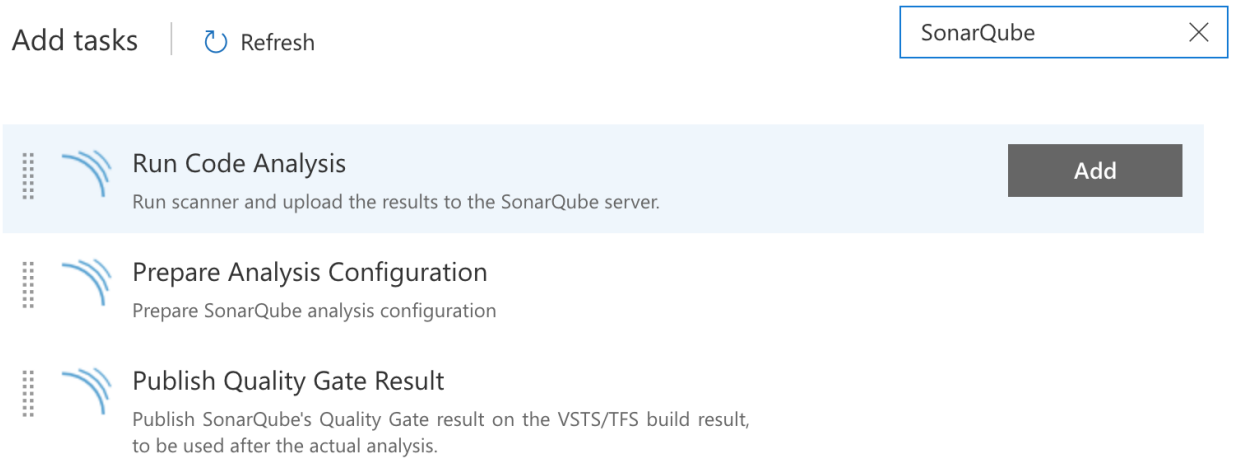
The first thing to do is to declare your SonarQube server as a service endpoint in your VSTS project settings. Read the detailed instructions on how to create a SonarQube endpoint.

Configure and run analyses

To analyse your projects, the extension provides 3 tasks that you will use in your build definitions:

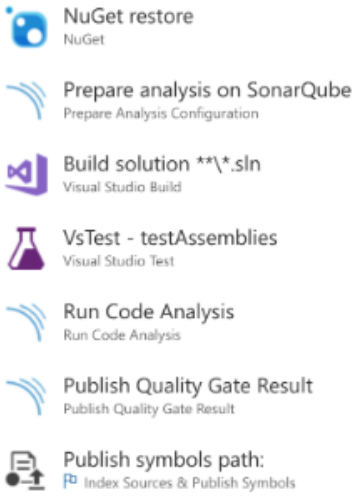
- **Prepare Analysis Configuration** task, to configure all the required settings before executing the build.
  - This task is mandatory.
  - In case of .NET solutions or Java projects, it helps to integrate seamlessly with MSBuild, Maven and Gradle tasks.
- **Run Code Analysis** task, to actually execute the analysis of the source code.
  - This task is not required for Maven or Gradle projects, because scanner will be run as part of the Maven/Gradle build.
- **Publish Quality Gate Result** task, to display the Quality Gate status in the build summary and give you a sense of whether the application is ready for production "quality-wise".
  - This tasks is optional.
  - It can increase significantly the overall build time, because it will poll the SonarQube server until the analysis is completed.

To find those tasks, you can type "SonarQube" to filter them:



Analysing a .NET solution

1. In your build definition, add:
   - At least **Prepare Analysis Configuration** task and **Run Code Analysis** task
   - Optionally **Publish Quality Gate Result** task
2. Reorder the tasks to respect the following order:
   - **Prepare Analysis Configuration** task before any **MSBuild** or **Visual Studio Build** task.
   - **Run Code Analysis** task after the **Visual Studio Test** task.
   - **Publish Quality Gate Result** task after the **Run Code Analysis** task



3. Click on the **Prepare Analysis Configuration** build step to configure it:
    a. The **SonarQube Server** section allows you to define the endpoint (i.e. SonarQube Server instance) to use.
    You can:
        - select an existing endpoint from the drop down list
        - add a new endpoint
        - manage existing endpoints

    b. Keep **Integrate with MSBuild** checked and specify at least the project key

        - **Project Key** - the unique project key in SonarQube
        - **Project Name** - the name of the project in SonarQube
        - **Project Version** - the version of the project in SonarQube
4. Click the **Visual Studio Test** task and check the **Code Coverage Enabled** checkbox to process the code coverage and have it imported into SonarQube. (*Optional but recommended*)

Once all this is done, you can trigger a build.

Analysing a Java project with Maven or Gradle

1. In your build definition, add:
    - At least **Prepare Analysis Configuration** task
    - Optionaly **Publish Quality Gate Result** task
2. Reorder the tasks to respect the following order:
    - **Prepare Analysis Configuration** task before the **Maven** or **Gradle** task.
    - **Publish Quality Gate Result** task after the **Maven** or **Gradle** task.
3. Click on the **Prepare Analysis Configuration** build step to configure it:
    a. Select the **SonarQube Server**
    b. Select **Integrate with Maven or Gradle**
4. On the Maven or Gradle task, in **Code Analysis**, check **Run SonarQube or SonarCloud Analysis**

Once all this is done, you can trigger a build.

Analysing other types of project

If you are not developing a .NET application or a Java project, here is the standard way to trigger an analysis:

1. In your build definition, add:
    - At least **Prepare Analysis Configuration** task and **Run Code Analysis** task
    - Optionaly **Publish Quality Gate Result** task
2. Reorder the tasks to respect the following order:
    - 1 - **Prepare Analysis Configuration** task.
    - 2 - **Run Code Analysis** task
    - 3 - **Publish Quality Gate Result** task.
3. Click on the **Prepare Analysis Configuration** build step to configure it:
    a. Select the **SonarQube Server**
    b. Select **Use standalone scanner**
    c. Then:

        i. Either the SonarQube properties are stored in the (standard) sonar-project.properties file in your SCM, and you just have to make sure that "Settings File" correctly points at it. This is the recommended way.
        ii. Or you don't have such a file in your SCM, and you can click on **Manually provide configuration** to specify it within your build definition. This is not recommended because it's less portable.

Once all this is done, you can trigger a build.

Branch and Pull Request

Branch analysis

Starting with SonarQube 7.2, when a build is run on a branch of your project, the extension automatically configures the analysis to be pushed to the relevant project branch on SonarQube. The same build definition can apply to all your branches, whatever type of Git repository you are analyzing,

If you are working with branches on TFVC projects or with SonarQube 6.7 LTS, you still need to manually specify the branch to be used on SonarQube: in **Prepare Analysis Configuration** task, in the **Additional Properties**, you need to set **sonar.branch.name**.

Pull request analysis

SonarQube 7.2+ can analyze the code of the new features and annotate your pull requests in TFS with comments to highlight issues that were found.

Pull request analysis is supported for any type of Git repositories. To activate it:

1. In the **Branch policies** page of your main development branches (e.g. "master"), add a build policy that runs your build definition
2. Create a TFS token with "Code (read and write)" scope
3. In SonarQube, in the "Administration > General Settings > Pull Requests" page, set this token in the "VSTS/TFS" section

Next time some code is pushed in the branch of a pull request, the build definition will execute a scan on the code and publish the results in SonarQube which will decorate the pull request in TFS.

On previous versions of the extension (3.0), this feature is available only for .NET solutions built and analysed with MSBuild. See the old documentation.

SonarCloud User?

If you are analyzing your project on SonarCloud, you should install the following VSTS extension: https://marketplace.visualstudio.com/items?itemName=SonarSource.sonarcloud

On top of all the features described in this page, it provides additional features to make the user experience even better when using SonarCloud:

- A single **SonarCloud dedicated end-point** allows to test connection to the service
- The **Prepare Analysis Configuration** task has built-in support for organizations and suggests the organizations you are member of

To get more info about how to use this extension, you should have a look at SonarCloud help: https://sonarcloud.io/documentation/integrations/vsts

FAQ

Is it possible to trigger analyses on Linux or macOS agents?

Starting with version 4.0 of the SonarQube task (1.0 for the SonarCloud extension), this is possible since the extension has been fully rewritten in Node.js, and the mono dependency was dropped in version 4.3 (1.3 for the SonarCloud extension).

This is not possible with previous versions of the extension.

How do I break the build base on the quality gate status?

This is not possible with the new version of the extension (4.0) if you are using the most up-to-date versions of the tasks. We believe that breaking a CI build is not the right approach. Instead, we are providing pull request decoration (to make sure that one does not introduce issues when merging his/her code) and we'll soon add a way to check the quality gate as part of a Release process.