sonarqube Docs 7.9

Search...

Architecture and Integration
Requirements
Setup and Upgrade
Analyzing Source Code
User Guide
Project Administration
Instance Administration
Extension Guide
Frequently Asked Questions

SonarScanner for Azure DevOps

By **SonarSource** - GNU LGPL 3 - **Issue Tracker** - **Source**
SonarScanner for Azure DevOps

The SonarQube extension for Azure DevOps Server makes it easy to integrate analysis into your build pipeline. The extension allows the analysis of all languages supported by SonarQube.

Compatibility

Version 4.x is compatible with:

- TFS 2017 Update 2+
- TFS 2018
- Azure DevOps Server 2019

The extension embeds its own version of the SonarScanner for MSBuild.

Installation

1. Install the extension from the marketplace.

If you are using Microsoft-hosted build agents then there is nothing else to install. The extension will work with all of the hosted agents (Windows, Linux, and MacOS).

2. If you are self-hosting the build agents make sure at least the minimal version of Java supported by SonarQube is installed. In addition, make sure the appropriate build tools are installed on the agent for the type of project e.g. .NET Framework v4.6+/NET Core 2.0+ if building using MSBuild, Maven for Java projects etc.

Configure

The first thing to do is to declare your SonarQube server as a service endpoint in your Azure DevOps project settings.

1. Open the Connections page in your Azure DevOps project: **Project Settings > Pipelines > Service Connections**.
2. Click on **New service connection** and choose **SonarQube**.
3. Specify a **Connection name**, the **Server URL** of your SonarQube Server (including the port if required) and the Authentication Token to use.

Each extension provides three tasks you will use in your build definitions to analyze your projects:

- **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.

SonarQube Community

sonarqube Docs 7.9

Search...

Architecture and Integration
Requirements
Setup and Upgrade
Analyzing Source Code
User Guide
Project Administration
Instance Administration
Extension Guide
Frequently Asked Questions

- It can significantly increase the overall build time because it will poll SonarQube until the analysis is complete. Omitting this task will not affect the analysis results on SonarQube - it simply means the Azure DevOps Build Summary page will not show the status of the analysis or a link to the project dashboard on SonarQube.

When creating a build definition you can filter the list of available tasks by typing "Sonar" to display only the relevant tasks.

Analyzing 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** tasks.
   - **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:
   - You must specify the service connection (i.e. SonarQube) to use. You can:
     - select an existing endpoint from the drop down list
     - add a new endpoint
     - manage existing endpoints
   - 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.

Analyzing a Java project with Maven or Gradle

1. In your build definition, add:
   - At least **Prepare Analysis Configuration** task
   - Optionally **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** task to configure it:
   - Select the **SonarQube Server**
   - 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.

SonarQube Community

sonarqube Docs 7.9

Search...

Architecture and Integration
Requirements
Setup and Upgrade
Analyzing Source Code
User Guide
Project Administration
Instance Administration
Extension Guide
Frequently Asked Questions

Analyzing a Visual C++ project

1. Make **SonarQube Build Wrapper** available on the build agent
   - Download and unzip **SonarQube Build Wrapper** on the build agent (see *Prerequisites* section of *C/C++/Objective-C* page). For the Microsoft-hosted build agent you will need to do it every time (as part of build definition), e.g. you can add **PowerShell script** task doing that. For the self-hosted build agent you can do the same either every build or only once (as part of manual setup of build agent). Example of PowerShell commands:

```
Invoke-WebRequest -Uri '<sonarqube or sonarcloud url>/static/cpp/build-wrapper-win-x86.zip' -OutFile 'build-wrapper.zip'
Expand-Archive -Path 'build-wrapper.zip' -DestinationPath '.'
```

2. In your build definition, add:
   - At least **Prepare Analysis Configuration** task, **Run Code Analysis** task and the **Command Line** task
   - Optionally **Publish Quality Gate Result** task
3. Reorder the tasks to respect the following order:
   - **Prepare Analysis Configuration** task before **Command Line** task.
   - **Run Code Analysis** task after the **Command Line** task.
   - **Publish Quality Gate Result** task after the **Run Code Analysis** task
4. On the **Command Line** task
   - Run **SonarQube Build Wrapper** executable. Pass in as the arguments (1) the output directory to which the Build Wrapper should write its results and (2) the command that runs the compilation of your project, e.g.

```
path/to/build-wrapper-win-x86-64.exe --out-dir <output directory> MSBuild.exe /t:Rebuild
```

5. Click on the **Prepare Analysis Configuration** task to configure it:
   - Select the **SonarQube Server**
   - In *Additional Properties* in the *Advanced* section, add the property `sonar.cfamily.build-wrapper-output` with the value of the directory you specified: `sonar.cfamily.build-wrapper-output=<output directory>`

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

Analysing other project types

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**
   2. **Run Code Analysis**
   3. **Publish Quality Gate Result**
3. Click on the **Prepare Analysis Configuration** task to configure it:
   - Select the **SonarQube Server**
   - Select **Use standalone scanner**
   - Then:

SonarQube Community

sonarqube Docs 7.9

Search...

Architecture and Integration
Requirements
Setup and Upgrade
Analyzing Source Code
User Guide
Project Administration
Instance Administration
Extension Guide
Frequently Asked Questions

- 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.
- 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 analysis

Branch and Pull Request analysis are available as part of Developer Edition and above

Branches

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 in 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, you still need to manually specify the branch to be used on SonarQube: in the **Prepare Analysis Configuration** task, in the **Additional Properties**, you need to set `sonar.branch.name`.

PRs

SonarQube 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 an Azure DevOps 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.

FAQ

Is it possible to trigger analyses on Linux or macOS agents?
This becomes possible from version 4.0 of the SonarQube task, in which the extension was fully rewritten in Node.js. The mono dependency was dropped in version 4.3.

This is not possible with previous versions of the extension.

How do I break the build based on the quality gate status?
This is not possible with the new version of the extension 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 issues aren't introduced at merge time) and we'll soon add a way to check the quality gate as part of a Release process.

SonarQube Community