SonarCloud - .NET Core

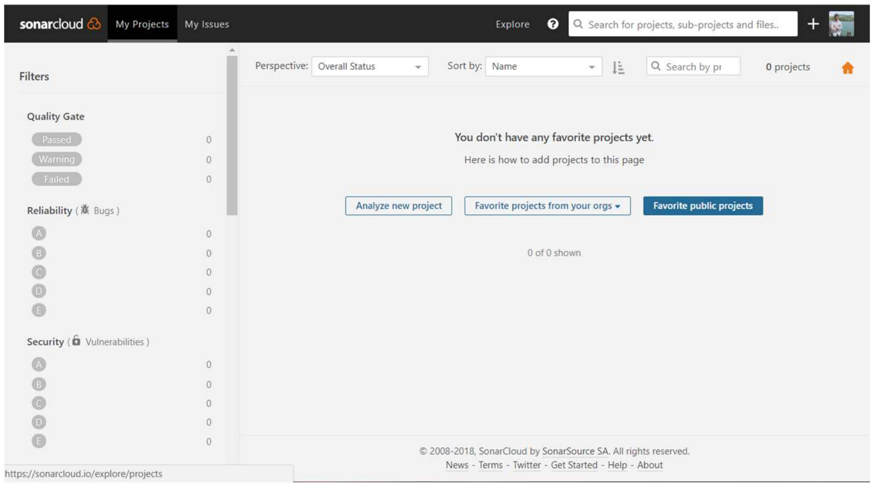
Contents

Static Code Analysis of .NET Core Projects with SonarCloud

https://dotnetthoughts.net/static-code-analysis-of-netcore-projects/

This post is about how to use SonarCloud application for running static code analysis in .NET Core projects. Static analysis is a way of automatically analysing code without executing it. SonarCloud is cloud offering of SonarQube app. It is Free for Open source projects.

For analysing first you need to create an account in sonarcloud.io. You can use GitHub / BitBucket / Microsoft Live Account to sign in. Once you sign in, you will see the sonarcloud dashboard.



You don't have any favorite projects yet.

Here is how to add projects to this page

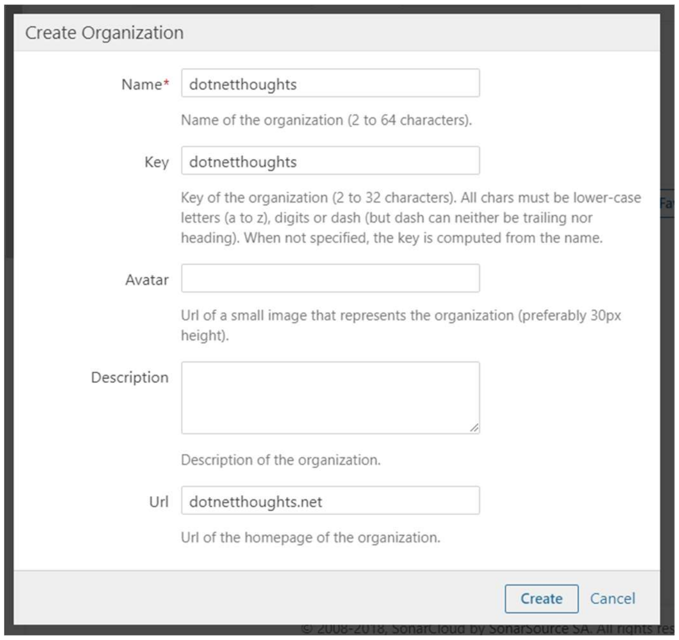
0 of 0 shown

© 2008-2018, SonarCloud by SonarSource SA. All rights reserved.

News - Terms - Twitter - Get Started - Help - About

https://sonarcloud.io/explore/projects

Next you need to create an Organization.

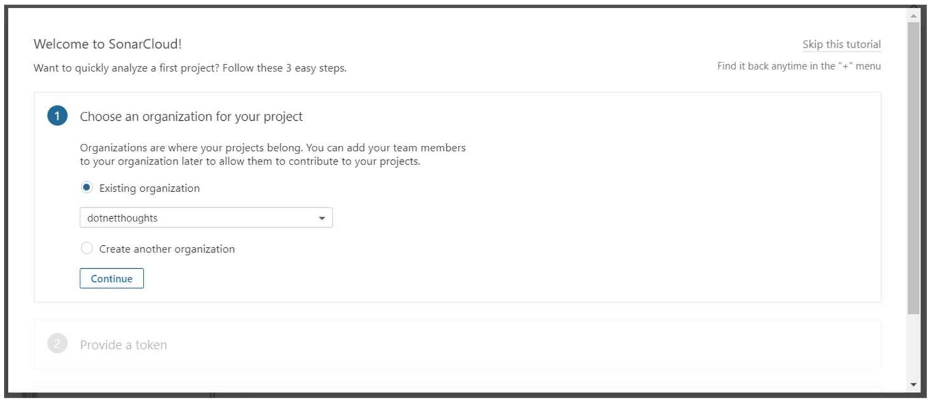


Create Organization

Name*	dotnetthoughts
	Name of the organization (2 to 64 characters).
Key	dotnetthoughts
	Key of the organization (2 to 32 characters). All chars must be lower-case letters (a to z), digits or dash (but dash can neither be trailing nor heading). When not specified, the key is computed from the name.
Avatar	
	Url of a small image that represents the organization (preferably 30px height).
Description	
	Description of the organization.
Url	dotnetthoughts.net
	Url of the homepage of the organization.

Once you created the organization, you can create a new project to analyse. You can click on the Analyse New Project button, which will show a wizard. In the first screen you need to select the organization.

SonarCloud - .NET Core



Welcome to SonarCloud!

Want to quickly analyze a first project? Follow these 3 easy steps.

Skip this tutorial

Find it back anytime in the "+" menu

1 Choose an organization for your project

Organizations are where your projects belong. You can add your team members to your organization later to allow them to contribute to your projects.

[✓ Existing organization]

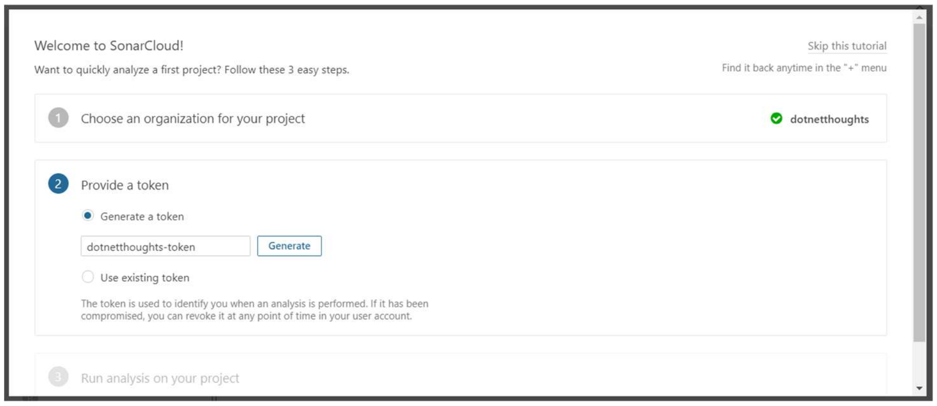
dotnetthoughts

Create another organization

Continue

2 Provide a token

Next, you need to generate token, which will be used to run the code analysis, make sure you store it safe.



Welcome to SonarCloud!

Want to quickly analyze a first project? Follow these 3 easy steps.

Skip this tutorial

Find it back anytime in the "+" menu

1 Choose an organization for your project ✓ dotnetthoughts

2 Provide a token

[✓ Generate a token]

dotnetthoughts-token

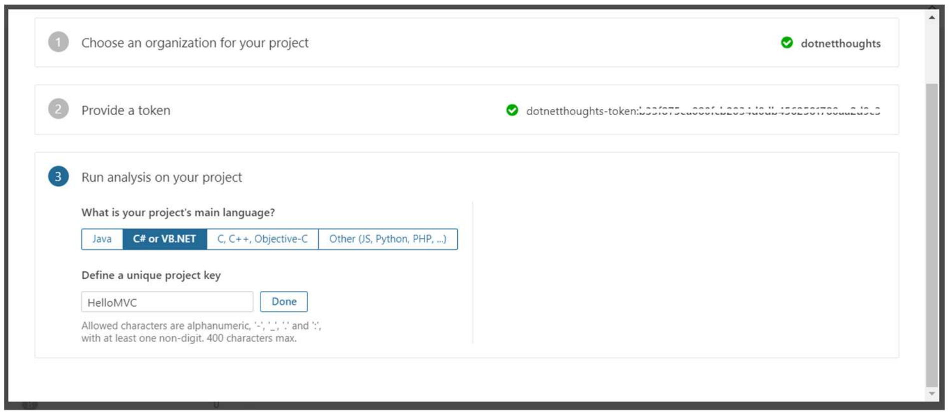
Generate

Use existing token

The token is used to identify you when an analysis is performed. If it has been compromised, you can revoke it at any point of time in your user account.

3 Run analysis on your project

And in the third screen, you need select the target language and provide a project key.



1 Choose an organization for your project ✓ dotnetthoughts

2 Provide a token ✓ dotnetthoughts-token:[redacted]

3 Run analysis on your project

What is your project's main language?

Java | [✓ C# or VB.NET] | C, C++, Objective-C | Other (JS, Python, PHP, ...)

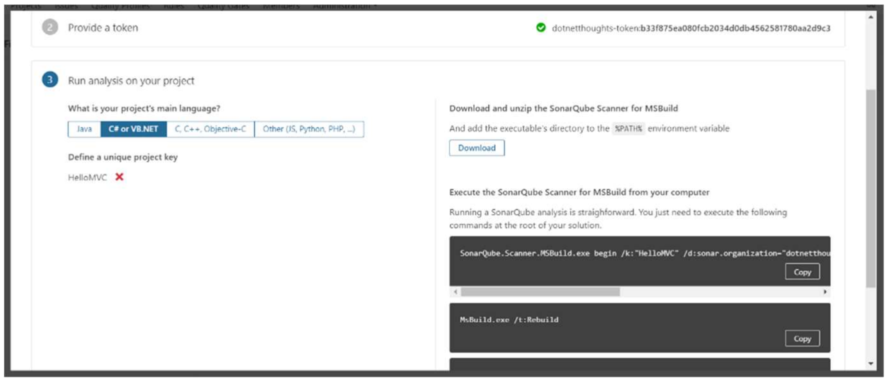
Define a unique project key

HelloMVC

Done

Allowed characters are alphanumeric, '-', '_', '.' and ':', with at least one non-digit. 400 characters max.

Clicking on Done, SonarCloud will show the steps to run code analysis using MSBuild.



dotnetthoughts-token:b33f875ea080fcb2034d0db4562581780aa2d9c3

What is your project's main language? C# or VB.NET

Define a unique project key

HelloMVC

Download and unzip the SonarQube Scanner for MSBuild

And add the executable's directory to the `%PATH%` environment variable

Execute the SonarQube Scanner for MSBuild from your computer

Running a SonarQube analysis is straighforward. You just need to execute the following commands at the root of your solution.

```
SonarQube.Scanner.MSBuild.exe begin /k:"HelloMVC" /d:sonar.organization="dotnetthou
```

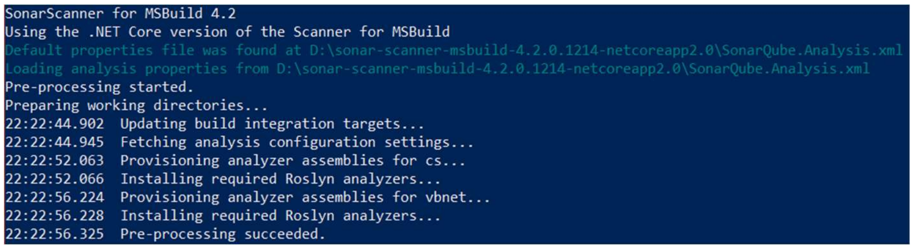
```
MsBuild.exe /t:Rebuild
```

Once you completed it, next you need to download the sonar scanner for MS Build.

Next you need to create the .NET Core application to run the analysis. So first you need to create a ASP.NET Core MVC project using `dotnet new mvc`. Then you need to create a sln file using `dotnet new sln` command. Next you need to add the MVC project to the solution. You can do it using `dotnet sln add HelloMVC.csproj`. Now you're ready with your project. Next you need to enable scanner. To do that first, you need to run the following command.

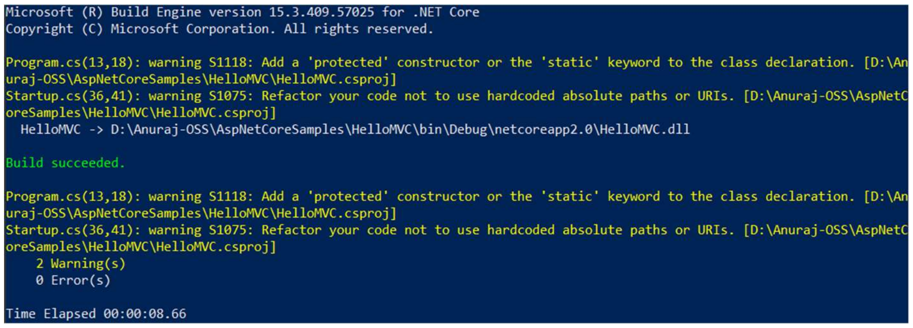
```
dotnet "D:\sonar-scanner-msbuild-4.2.0.1214-netcoreapp2.0\SonarScanner.MSBuild.dll" b
egin /k:"HelloMVC" /d:sonar.organization="dotnetthoughts" /d:sonar.host.url="https://
sonarcloud.io" /d:sonar.login="73fd8bc705804e8688b797f0e70dc6d70aa2d9c3"
```

Once you execute the command you will be able to see something like this on the screen.



```
SonarScanner for MSBuild 4.2
Using the .NET Core version of the Scanner for MSBuild
Default properties file was found at D:\sonar-scanner-msbuild-4.2.0.1214-netcoreapp2.0\SonarQube.Analysis.xml
Loading analysis properties from D:\sonar-scanner-msbuild-4.2.0.1214-netcoreapp2.0\SonarQube.Analysis.xml
Pre-processing started.
Preparing working directories...
22:22:44.902  Updating build integration targets...
22:22:44.945  Fetching analysis configuration settings...
22:22:52.063  Provisioning analyzer assemblies for cs...
22:22:52.066  Installing required Roslyn analyzers...
22:22:56.224  Provisioning analyzer assemblies for vbnet...
22:22:56.228  Installing required Roslyn analyzers...
22:22:56.325  Pre-processing succeeded.
```

Next you need to build the application using `dotnet build` command. You will be able to see some warnings in the screen.

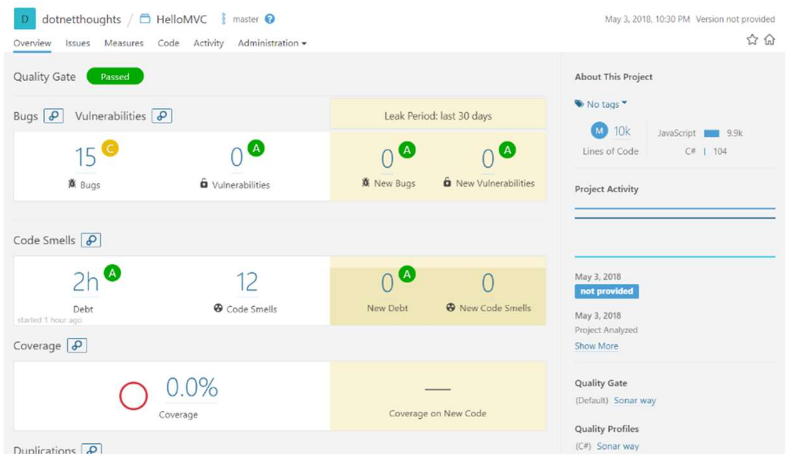


```
Microsoft (R) Build Engine version 15.3.409.57025 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

Program.cs(13,18): warning S1118: Add a 'protected' constructor or the 'static' keyword to the class declaration. [D:\An
uraj-OSS\AspNetCoreSamples\HelloMVC\HelloMVC.csproj]
Startup.cs(36,41): warning S1075: Refactor your code not to use hardcoded absolute paths or URIs. [D:\Anuraj-OSS\AspNetC
oreSamples\HelloMVC\HelloMVC.csproj]
  HelloMVC -> D:\Anuraj-OSS\AspNetCoreSamples\HelloMVC\bin\Debug\netcoreapp2.0\HelloMVC.dll

Build succeeded.

Program.cs(13,18): warning S1118: Add a 'protected' constructor or the 'static' keyword to the class declaration. [D:\An
uraj-OSS\AspNetCoreSamples\HelloMVC\HelloMVC.csproj]
Startup.cs(36,41): warning S1075: Refactor your code not to use hardcoded absolute paths or URIs. [D:\Anuraj-OSS\AspNetC
oreSamples\HelloMVC\HelloMVC.csproj]
    2 Warning(s)
    0 Error(s)

Time Elapsed 00:00:08.66
```

To finish you need to run the `end` command similar to `begin` command.

```
dotnet "D:\sonar-scanner-msbuild-4.2.0.1214-netcoreapp2.0\SonarScanner.MSBuild.dll" e
nd /d:sonar.login="73fd8bc705804e8688b797f0e70dc6d70aa2d9c3"
```

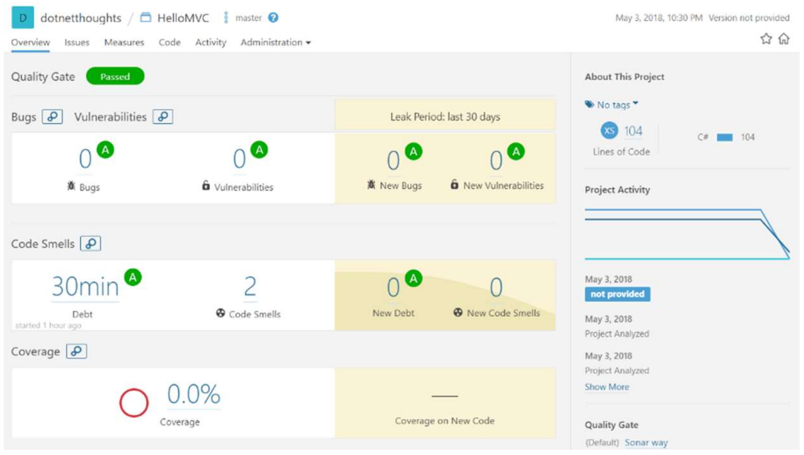
This will start the analysis and upload the results to Sonarcloud. Next open your SonarCloud dashboard, click on the project, then you will be able to see the results like this.



If you notice, SonarCloud is showing 15 bugs, but if you look into the details you will be able to see most of these issues reported is from JavaScript libraries, like Bootstrap or JQuery. In ideal scenario, you don't need to run code analysis on this library files. So you need to exclude the libraries. You can do this using SonarQube exclusion filters like this.

```
dotnet "D:\sonar-scanner-msbuild-4.2.0.1214-netcoreapp2.0\SonarScanner.MSBuild.dll" b
egin /k:"HelloMVC" /d:sonar.organization="dotnetthoughts" /d:sonar.host.url="https://
sonarcloud.io" /d:sonar.login="73fd8bc705804e8688b797f0e70dc6d70aa2d9c3" /d:sonar.exc
lusions="/wwwroot/lib/**"
```

Next build using dotnet build and end the scanning. It will update the dashboard like this.



Happy Programming :)

SonarScanner for MSBuild

https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-msbuild/

Download SonarScanner for MSBuild 4.6.2.2108 - Compatible with SonarQube 6.7+ (LTS)
By **SonarSource** – GNU LGPL 3 – **Issue Tracker** – **Source**
.NET Framework 4.6+ | **.NET Core 2.0+** | **.NET Core Global Tool**

The SonarScanner for MSBuild is the recommended way to launch an analysis for projects/solutions using MSBuild or dotnet command as a build tool. It is the result of a collaboration between SonarSource and Microsoft.

SonarScanner for MSBuild is distributed as a standalone command line executable, as a extension for Azure DevOps Server, and as a plugin for Jenkins.

It supports .Net Core multi-platform projects and it can be used on non-Windows platforms.

Prerequisites

- At least the minimal version of Java supported by your SonarQube server
- The SDK corresponding to your build system:
    - .NET Framework v4.6+ - either Build Tools for Visual Studio 2015 Update 3 or the Build Tools for Visual Studio 2017
    - .NET Core 2.0 - .NET Core SDK 2.0 (for .NET Core version of the scanner)
    - .NET Core 2.1 - NET Core SDK 2.1.3 (for .NET Core version of the scanner or if you plan to use .NET Core Global Tool

Installation

SonarScanner for MSBuild for .NET Framework or .NET Core

- Expand the downloaded file into the directory of your choice. We'll refer to it as $install_directory in the next steps.
    - On Windows, you might need to unblock the ZIP file first (Right click on file > Properties > Unblock).
    - On Linux/OSX you may need to set execute permissions on the files in $install_directory/sonar-scanner-(version)/bin.
- Uncomment, and update the global settings to point to your SonarQube server by editing $install_directory/SonarQube.Analysis.xml. Values set in this file will be applied to all analyses of all projects unless overwritten locally.
  Consider setting file system permissions to restrict access to this file.:

```
• <SonarQubeAnalysisProperties  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://www.sonarsource.com/msbuild/integration/2015/1">
• <Property Name="sonar.host.url">http://localhost:9000</Property>
```

- `<Property Name="sonar.login">[my-user-token]</Property>`
- `</SonarQubeAnalysisProperties>`

- Add `$install_directory` to your PATH environment variable.

Installation of the SonarScanner for MSBuild .NET Core Global Tool

```
dotnet tool install --global dotnet-sonarscanner --version 4.3.1
```

The *--version* argument is optional. If it is omitted the latest version will be installed.

On Linux/OSX, if your SonarQube server is secured:

- Copy the server's CA certs to `/usr/local/share/ca-certificates`
- Run `sudo update-ca-certificates`

Use

There are two versions of the SonarScanner for MSBuild.

The first version is based on the “classic” .NET Framework. To use it, execute the following commands from the root folder of your project:

```
SonarScanner.MSBuild.exe begin /k:"project-key"
MSBuild.exe <path to solution.sln> /t:Rebuild
SonarScanner.MSBuild.exe end
```

Note: On Mac OS or Linux, you can also use `mono <path to SonarScanner.MSBuild.exe>`.

The second version is based on .NET Core which has a very similar usage:

```
dotnet <path to SonarScanner.MSBuild.dll> begin /k:"project-key"
dotnet build <path to solution.sln>
dotnet <path to SonarScanner.MSBuild.dll> end
```

The .NET Core version can also be used as a .NET Core Global Tool. After installing the Scanner as a global tool as described above it can be invoked as follows:

```
dotnet sonarscanner begin /k:"project-key"
dotnet build <path to solution.sln>
dotnet sonarscanner end
```

Same as above, if you are targetting a SonarCloud project, will have to add both the organization and a login for authentication.

Notes:

- The .NET Core version of the scanner does not support TFS XAML builds. Apart from that, the two versions of scanner have the same capabilities and command line arguments.

- Single .NET Core project files (csproj or vbproj) could be built and successfully analyzed only if a `<ProjectGuid>unique guid</ProjectGuid>` element is added in the csproj or vbproj XML. The `<ProjectGuid>` element is not required if you build a solution (sln) containing that project.

Analysis steps

Begin

The begin step is executed when you add the `begin` command line argument. It hooks into the MSBuild pipeline, downloads SonarQube quality profiles and settings and prepares your project for the analysis.

Command Line Parameters:

Parameter	**Description**
`/k:<project-key>`	[required] Specifies the key of the analyzed project in SonarQube
`/n:<project name>`	[optional] Specifies the name of the analyzed project in SonarQube. Adding this argument will overwrite the project name in SonarQube if it already exists.
`/v:<version>`	[recommended] Specifies the version of your project.

`/d:sonar.login=<username> or <token>`| [optional] Specifies the username or access token to authenticate with to SonarQube. If this argument is added to the begin step, it must also be added on the end step. `/d:sonar.password=<password>`|[optional] Specifies the password for the SonarQube username in the `sonar.login` argument. This argument is not needed if you use authentication token. If this argument is added to the begin step, it must also be added on the end step. `/d:sonar.verbose=true`|[optional] Sets the logging verbosity to detailed. Add this argument before sending logs for troubleshooting. `/d:<analysis-parameter>=<value>`|[optional] Specifies an additional SonarQube analysis parameter, you can add this argument multiple times.

For detailed information about all available parameters, see Analysis Parameters.

The "begin" step will modify your build like this:

- the active `CodeAnalysisRuleSet` will be updated to match the SonarQube quality profile
- `WarningsAsErrors` will be turned off

If your build process cannot tolerate these changes we recommend creating a second build job for SonarQube analysis.

Build

Between the `begin` and `end` steps, you need to build your project, execute tests and generate code coverage data. This part is specific to your needs and it is not detailed here.

End

The end step is executed when you add the "end" command line argument. It cleans the MSBuild hooks, collects the analysis data generated by the build, the test results, the code coverage and then uploads everything to SonarQube.

There are only two additional arguments that are allowed for the end step:

Parameter	Description
`/d:sonar.login=<username> or <token>`	[optional] This argument is required if it was added to the begin step.
`/d:sonar.password=<password>`	[optional] This argument is required if it was added to the begin step and you are not using an authentication token.

Known Limitations

- MSBuild versions older than 14 are not supported.
- Web Application projects are supported. Legacy Web Site projects are not.
- Projects targeting multiple frameworks and using preprocessor directives could have slightly inaccurate metrics (lines of code, complexity, etc.) because the metrics are calculated only from the first of the built targets.

Excluding projects from analysis

Some project types, such as Microsoft Fakes, are automatically excluded from analysis. To manually exclude a different type of project from the analysis, place the following in its .xxproj file.

```
<!-- in .csproj -->
<PropertyGroup>
  <!-- Exclude the project from analysis -->
  <SonarQubeExclude>true</SonarQubeExclude>
</PropertyGroup>
```

Advanced topics

Analyzing MSBuild 12 projects with MSBuild 14

The Sonar Scanner for MSBuild requires your project to be built with MSBuild 14.0. We recommend installing Visual Studio 2015 update 3 or later on the analysis machine in order to benefit from the integration and features provided with the Visual Studio ecosystem (VSTest, MSTest unit tests, etc.).

Projects targeting older versions of the .NET Framework can be built using MSBuild 14.0 by setting the "TargetFrameworkVersion" MSBuild property as documented by Microsoft:

- How to: Target a Version of the .NET Framework
- MSBuild Target Framework and Target Platform

If you do not want to switch your production build to MSBuild 14.0, you can set up a separate build dedicated to the SonarQube analysis.

Detection of Test Projects

SonarQube analyzes test projects differently from non-test projects, so it is important to correctly classify test projects.

By default, the SonarQube Scanner for MSBuild will detect as test project:

. MSTest unit test projects, thanks to the presence of a well-known project type GUID in .csproj file of such projects.

1. Projects with names ending in "Test" or "Tests". This behavior can be changed by providing the parameter `sonar.msbuild.testProjectPattern` to the begin step (regex follows .NET Regular Expression in a case-sensitive way with the default value `.*Tests?\.(cs|vb)proj$`). This regex is applied against the fullname of the `.csproj` or `.vbproj` which is why it's recommended to keep at the end of your custom regex `\.(cs|vb)proj$`. To manually classify a project as a test project, mark it with `<SonarQubeTestProject>true</SonarQubeTestProject>`:

```
<!-- in .csproj -->
<PropertyGroup>
<!-- Mark the project as being a test project -->
<SonarQubeTestProject>true</SonarQubeTestProject>
</PropertyGroup>
```

Per-project analysis parameters Some analysis parameters can be set for a single MSBuild project by adding them to its .csproj file.

```
<!-- in .csproj -->
<ItemGroup>
  <SonarQubeSetting Include="sonar.stylecop.projectFilePath">
    <Value>$(MSBuildProjectFullPath)</Value>
  </SonarQubeSetting>
</ItemGroup>
```

Concurrent Analyses on the Same Build Machine

Concurrent analyses (i.e. parallel analysis of two solutions on the same build machine using a unique service account) are not supported by default by the Scanner for MSBuild. You can enable it as follows:

1. Locate the folder containing the Scanner for MSBuild
2. Go in the `Targets` folder and copy the folder `SonarQube.Integration.ImportBefore.targets`
3. Paste it under your build tool global `ImportBefore` folder (if the folder doesn't exist, create it).

  - For MSBuild, the path is `<MSBUILD_INSTALL_DIR>\<Version>\Microsoft.Common.targets\ImportBefore` where <MSBUILD*INSTALL*DIR> is: *For v14, default path is: `C:\Program Files (x86)\MSBuild\14.0\Microsoft.Common.Targets\ImportBefore`
    - For v15, default path is: `C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\MSBuild\15.0\Microsoft.Common.targets\ImportBefore` (for VS Community Edition)
    - For v16, default path is: `C:\Program Files (x86)\Microsoft Visual Studio\2019\Enterprise\MSBuild\Current\Microsoft.Common.targets` (for VS Community Edition)
  - For dotnet, the path is `<DOTNET_SDK_INSTALL_DIR>\15.0\Microsoft.Common.targets\ImportBefore` where `<DOTNET_SDK_INSTALL_DIR>` can be found using the `dotnet --info` and looking for the Base Path property.

The performance impact of this global installation for projects that aren't analyzed is negligible as this target is only a bootstrapper and will bail out nearly instantaneously when the `.sonarqube`folder is not found under the folder being built.

Using SonarScanner for MSBuild with a Proxy

On build machines that connect to the Internet through a proxy server you might experience difficulties connecting to SonarQube. To instruct the Java VM to use the system proxy settings, you need to set the following environment variable before running the SonarScanner for MSBuild:

```
SONAR_SCANNER_OPTS = "-Djava.net.useSystemProxies=true"
```

To instruct the Java VM to use specific proxy settings or when there is no system-wide configuration use the following value:

```
SONAR_SCANNER_OPTS = "-Dhttp.proxyHost=yourProxyHost -Dhttp.proxyPort=yourProxyPort"
```

Where *yourProxyHost* and *yourProxyPort* are the hostname and the port of your proxy server. There are additional proxy settings for https, authentication and exclusions that could be passed to the Java VM.