Configure Sonar for C/C++

Depending on the setup of the repo, scanning a C/C++ project may involve a bit more setup and configuration, and unlike scripting languages, Sonar requires that the code to be analysed also be compiled by the *build wrapper* (a Sonar data collector). This page contains some extra details that might be useful for configuring a C/C++ repo for scanning.

Overall Mechanics

The LF Jenkins job "gerrit-cmake-sonarqube" will preform the scanning work and specifically does the following:

1 Create the directory "build" at the repo root
1 Run CMake in that directory
1 Run make

Parameters are supplied to the gerrit job via the usual YAML file in the ci-management repo. The following is an example set of parameters for the scanner job.

Sample sonar analysis job setup

```
    - project:
      <<: *mc_common
      name: ric-app-mc-sonarqube
      project-name: ric-app-mc
      cmake-opts: ""
      make-opts: test ARGS=-V
      sonar-project-file: ""
      sonar-properties: |
          sonar.login={sonarcloud_api_token}
          sonar.projectKey={sonarcloud_project_organization}_{project-name}
          sonar.projectName={project-name}
          sonar.organization={sonarcloud_project_organization}
          sonar.build.sourceEncoding=UTF-8
          sonar.sources=sidecars/listener/src
          sonar.cfamily.build-wrapper-output=$WORKSPACE/bw-output
          sonar.cfamily.gcov.reportsPath=sidecars/listener
          sonar.cfamily.threads=2
      jobs:
        - gerrit-cmake-sonarqube
```

The cmake-opts allows options to be provided on what otherwise would be the command: cmake .. executed in the build directory. For the example above there are none, but for the C++ framework this likely sets the package type to build (e.g. -DDEV_PKG=1) as the unit tests need that.

The "make-opts" are used when running make in the build directory. These may be empty, but are likely going to be a minimum of "test" or whatever make rule(s) is/are created by the CMake process. In the above example the "ARGS=-V" option is passed to make such that the full output from the unit test script is written to stdout/err; normally CMake sets up tests to run very quietly.

Sonar Properties

Rather than providing a Sonar configuration file at the repo root, properties are passed via the list in the yaml. Some
should be included as they are defined in the previous example (e.g. sonar.login), while others can/should be configured for the test. The parameters discussed below are the ones which seem most useful and/or important with a description of the full set of properties provided by [4].

sonar.sources
This is the directory where the sources exist for analysis.
This may be a comma separated list of paths.

sonar.exclusions
This provides a list of directories not to run analysis on. According to someone from the Sonar dev team this is hard[1]:

"Correctly setting exclusions from properties is difficult to get right, which is why it's not documented. Instead, you should use the UI to set your exclusions."

Unfortunately, we don't have access to the UI, so we must set exclusions from the properties list. (The poster was also slammed as "hard" isn't an excuse for not providing documentation for the property.)

This property accepts a comma separated list of directory paths which will not be included in the analysis. Generally this is needed only to exclude generated code, or test code which exists in a directory below one of the sources directories.

Sonar doesn't use "normal" regular expressions, or even shell wild carding, when defining file patterns for inclusion/exclusion. It seems that the "ant" (java's remake of good ole make?) defined it's own syntax which is both less than obvious and strange. [2] and [3] might help to understand.

sonar.coverage.exclusions
This provides a list of directories not to run coverage on. If more than one directory is given they are comma separated.

C/C++ Analysis

The "cfamily" scanner has its own set of parameters which affect the behaviour [5]. The page also contains links to other supported languages. The ones which seem most important are described below.

sonar.cfamily.gcov.reportsPath
This is the directory path where the .gcov (C/C++ coverage) will exist following the unit test execution. The jobs will look only at the .gcov files in this directory.

sonar.cfamily.build-wrapper-output
The scanning process will create this directory and place output (a log and json file) into it. For the most part it's location is unimportant, but it must be along a path which is writable at the time of job execution, so using $WOKSPACE as the base is advisable.

The Wrapper Must "See" The Build

The "make" executed by the Jenkins job (step three listed earlier) is actually executed by a "build wrapper" which is a Sonar "black box" process. One problem, which took a few days to solve, was the need for the wrapper to "see" a build in the source directory. To better organise the code, the test scripts and code were moved from the main source directory to a directory which was outside of the source directory root and thus excluding it from analysis (desired). When this was done, there was no need to build the code in the source directory, and the wrapper stopped identifying code to analyse.

The takeaway from this was that either the make that is executed in the build directory must build in the main source directory. For packages like the C++ framework, the make options cause both the package and the unit test rules to trigger, but for the project where the problem was encountered the make rule only triggered the unit test script. So, for some projects this might not be encountered as it happens "naturally," but for other projects the build in the source directory might need to be explicitly forced by the unit test script, or other mechanism.

References

[1] Stackoverflow
"SonarQube: sonar.exclusions parameter cannot exclude a folder"
https://stackoverflow.com/questions/48585622/sonarqube-sonar-exclusions-parameter-cannot-exclude-a-folder

[2] Sonar Documentation: "Narrowing the Focus"
https://docs.sonarqube.org/latest/project-administration/narrowing-the-focus/

[3] Stackoverflow
"How do I use Nant/Ant naming patterns?"
https://stackoverflow.com/questions/69835/how-do-i-use-nant-ant-naming-patterns/86915#86915

[4] Sonar Documentation: "Analysis Parameters"
https://docs.sonarqube.org/latest/analysis/analysis-parameters/

[5] Sonar Documentation: "C/C++/Objective-C"
https://docs.sonarqube.org/latest/analysis/languages/cfamily/