Analyzing your code with CodeQL queries

In this article

You can run queries against a CodeQL database extracted from a codebase.

About analyzing databases with the CodeQL CLI

To analyze a codebase, you run queries against a CodeQL database extracted from the code. CodeQL analyses produce results that can be uploaded to GitHub Enterprise Cloud to generate code scanning alerts.

Prerequisites

Before starting an analysis you must:

- Set up the CodeQL CLI to run commands locally.
- Create a CodeQL database for the source code you want to analyze.

The simplest way to run `codeql database analyze` is using the standard queries included in the CodeQL CLI bundle.

Running `codeql database analyze`

When you run `database analyze`, it:

1. Optionally downloads any referenced CodeQL packages that are not available locally.
2. Executes one or more query files, by running them over a CodeQL database.

3. Interprets the results, based on certain query metadata, so that alerts can be displayed in the correct location in the source code.
4. Reports the results of any diagnostic and summary queries to standard output.

You can analyze a database by running the following command:

```
codeql database analyze <database> --format=<format> --output=<output> <query-specifiers>...
```

Note: If you analyze more than one CodeQL database for a single commit, you must specify a SARIF category for each set of results generated by this command. When you upload the results to GitHub Enterprise Cloud, code scanning uses this category to store the results for each language separately. If you forget to do this, each upload overwrites the previous results.

```
codeql database analyze <database> --format=<format> \
    --sarif-category=<language-specifier> --output=<output> \
    <packs,queries>
```

You must specify `<database>`, `--format`, and `--output`. You can specify additional options depending on what analysis you want to do.

Option	Required	Usage
`<database>`	✓	Specify the path for the directory that contains the CodeQL database to analyze.
`<packs,queries>`	✗	Specify CodeQL packs or queries to run. To run the standard queries used for code scanning, omit this parameter. To see the other query suites included in the CodeQL CLI bundle, look in `/<extraction-root>/qlpacks/codeql/<language>-queries/codeql-suites`. For information about creating your own query suite, see Creating CodeQL query suites in the documentation for the CodeQL CLI.
`--format`	✓	Specify the format for the results file generated during analysis. A number of different formats are supported, including CSV, SARIF, and graph formats. For upload to GitHub this should be: `sarif-latest`. For more information, see "SARIF support for code scanning."
`--output`	✓	Specify the location where you want to save the SARIF results file, including the desired filename with the `.sarif` extension.
`--sarif-category`	?	Optional for single database analysis. Required to define the

		language when you analyze multiple databases for a single commit in a repository. Specify a category to include in the SARIF results file for this analysis. A category is used to distinguish multiple analyses for the same tool and commit, but performed on different languages or different parts of the code.
`--sarif-add-baseline-file-info`	×	**Recommended.** Use to submit file coverage information to the tool status page. For more information, see "About the tool status page for code scanning."
`--sarif-add-query-help`	×	Use if you want to include any available markdown-rendered query help for custom queries used in your analysis. Any query help for custom queries included in the SARIF output will be displayed in the code scanning UI if the relevant query generates an alert. For more information, see "Using custom queries with the CodeQL CLI."
`<packs>`	×	Use if you want to include CodeQL query packs in your analysis. For more information, see "Downloading and using CodeQL packs."
`--download`	×	Use if some of your CodeQL query packs are not yet on disk and need to be downloaded before running queries.
`--threads`	×	Use if you want to use more than one thread to run queries. The default value is `1`. You can specify more threads to speed up query execution. To set the number of threads to the number of logical processors, specify `0`.
`--verbose`	×	Use to get more detailed information about the analysis process and diagnostic data from the database creation process.

Upgrading databases

For databases that were created by CodeQL CLI v2.3.3 or earlier, you will need to explicitly upgrade the database before you can run an analysis with a newer version of the CodeQL CLI. If this step is necessary, then you will see a message telling you that your database needs to be upgraded when you run `database analyze`.

For databases that were created by CodeQL CLI v2.3.4 or later, the CLI will implicitly run any required upgrades. Explicitly running the upgrade command is not necessary.

For full details of all the options you can use when analyzing databases, see "database analyze."

Basic example of analyzing a CodeQL database

This example analyzes a CodeQL database stored at `/codeql-dbs/example-repo` and saves the results as a SARIF file: `/temp/example-repo-js.sarif`. It uses `--sarif-category` to include extra information in the SARIF file that identifies the results as JavaScript. This is essential when you have more than one CodeQL database to analyze for a single commit in a repository.

```
$ codeql database analyze /codeql-dbs/example-repo \
    javascript-code-scanning.qls --sarif-category=javascript-typescript \
    --format=sarif-latest --output=/temp/example-repo-js.sarif

> Running queries.
> Compiling query plan for /codeql-home/codeql/qlpacks/codeql-
javascript/AngularJS/DisablingSce.ql.
...
> Shutting down query evaluator.
> Interpreting results.
```

Adding file coverage information to your results for monitoring

You can optionally submit file coverage information to GitHub Enterprise Cloud for display on the tool status page for code scanning. For more information about file coverage information, see "About the tool status page for code scanning."

To include file coverage information with your code scanning results, add the `--sarif-add-baseline-file-info` flag to the `codeql database analyze` invocation in your CI system, for example:

```
$ codeql database analyze /codeql-dbs/example-repo \
    javascript-code-scanning.qls --sarif-category=javascript-typescript \
    --sarif-add-baseline-file-info \ --format=sarif-latest \
    --output=/temp/example-repo-js.sarif
```

Examples of running database analyses

The following examples show how to run `database analyze` using CodeQL packs, and how to use a local checkout of the CodeQL repository. These examples assume your CodeQL databases have been created in a directory that is a sibling of your local copies of the CodeQL repository.

Running a CodeQL query pack

Note

The CodeQL package management functionality, including CodeQL packs, is currently available as a beta release and is subject to change. During the beta release, CodeQL packs are available only using GitHub Packages - the GitHub Container registry. To use this beta functionality, install the latest version of the CodeQL CLI bundle from: https://github.com/github/codeql-action/releases.

To run an existing CodeQL query pack from the GitHub Container registry, you can specify one or more pack names:

```
codeql database analyze <database> microsoft/coding-standards@1.0.0
github/security-queries --format=sarifv2.1.0 --output=query-results.sarif --
download
```

This command runs the default query suite of two CodeQL query packs: `microsoft/coding-standards` version 1.0.0 and the latest version of `github/security-queries` on the specified database. For further information about default suites, see "Publishing and using CodeQL packs."

The `--download` flag is optional. Using it will ensure the query pack is downloaded if it isn't yet available locally.

Running a single query

To run a single query over a CodeQL database for a JavaScript codebase, you could use the following command from the directory containing your database:

```
codeql database analyze --download <javascript-database> codeql/javascript-
queries:Declarations/UnusedVariable.ql --format=csv --output=js-analysis/js-
results.csv
```

This command runs a simple query that finds potential bugs related to unused variables, imports, functions, or classes—it is one of the JavaScript queries included in the CodeQL repository. You could run more than one query by specifying a space-separated list of similar paths.

The analysis generates a CSV file (`js-results.csv`) in a new directory (`js-analysis`).

Alternatively, if you have the CodeQL repository checked out, you can execute the same queries by specifying the path to the query directly:

```
codeql database analyze <javascript-database>
../ql/javascript/ql/src/Declarations/UnusedVariable.ql --format=csv --output=js-
analysis/js-results.csv
```

You can also run your own custom queries with the `database analyze` command. For more information about preparing your queries to use with the CodeQL CLI, see "Using custom queries with the CodeQL CLI."

Running all queries in a directory

You can run all the queries located in a directory by providing the directory path, rather than listing all the individual query files. Paths are searched recursively, so any queries contained in subfolders will also be executed.

Important

You should avoid specifying the root of a core CodeQL query pack when executing `database analyze` as it might contain some special queries that aren't designed to be used with the command. Rather, run the query pack to include the pack's default queries in the analysis, or run one of the code scanning query suites.

For example, to execute all Python queries contained in the `Functions` directory in the `codeql/python-queries` query pack you would run:

```
codeql database analyze <python-database> codeql/python-queries:Functions --
format=sarif-latest --output=python-analysis/python-results.sarif --download
```

Alternatively, if you have the CodeQL repository checked out, you can execute the same queries by specifying the path to the directory directly:

```
codeql database analyze <python-database> ../ql/python/ql/src/Functions/ --
format=sarif-latest --output=python-analysis/python-results.sarif
```

When the analysis has finished, a SARIF results file is generated. Specifying `--format=sarif-latest` ensures that the results are formatted according to the most recent SARIF specification supported by CodeQL.

Running a subset of queries in a CodeQL pack

If you are using CodeQL CLI v2.8.1 or later, you can include a path at the end of a pack specification to run a subset of queries inside the pack. This applies to any command that locates or runs queries within a pack.

The complete way to specify a set of queries is in the form `scope/name@range:path`, where:

- `scope/name` is the qualified name of a CodeQL pack.
- `range` is a semver range.
- `path` is a file system path to a single query, a directory containing queries, or a query suite file.

When you specify a `scope/name`, the `range` and `path` are optional. If you omit a `range` then the latest version of the specified pack is used. If you omit a `path` then the default query suite of the specified pack is used.

The `path` can be one of a `\*.ql` query file, a directory containing one or more queries, or a `.qls` query suite file. If you omit a pack name, then you must provide a `path`, which will be interpreted relative to the working directory of the current process.

If you specify a `scope/name` and `path`, then the `path` cannot be absolute. It is considered relative to the root of the CodeQL pack.

To analyze a database using all queries in the `experimental/Security` folder within the `codeql/cpp-queries` CodeQL pack you can use:

```
codeql database analyze --format=sarif-latest --output=results <db> \
    codeql/cpp-queries:experimental/Security
```

To run the `RedundantNullCheckParam.ql` query in the `codeql/cpp-queries` CodeQL pack use:

```
codeql database analyze --format=sarif-latest --output=results <db> \
    'codeql/cpp-queries:experimental/Likely Bugs/RedundantNullCheckParam.ql'
```

To analyze your database using the `cpp-security-and-quality.qls` query suite from a version of the `codeql/cpp-queries` CodeQL pack that is >= 0.0.3 and < 0.1.0 (the highest compatible version will be chosen) you can use:

```
codeql database analyze --format=sarif-latest --output=results <db> \
   'codeql/cpp-queries@~0.0.3:codeql-suites/cpp-security-and-quality.qls'
```

If you need to reference a query file, directory, or suite whose path contains a literal `@` or `:`, you can prefix the query specification with `path:` like so:

```
codeql database analyze --format=sarif-latest --output=results <db> \
    path:C:/Users/ci/workspace@2/security/query.ql
```

For more information about CodeQL packs, see Customizing analysis with CodeQL packs.

Running query suites

To run a query suite on a CodeQL database for a C/C++ codebase, you could use the following command from the directory containing your database:

```
codeql database analyze <cpp-database> codeql/cpp-queries:codeql-suites/cpp-code-scanning.qls --format=sarifv2.1.0 --output=cpp-results.sarif --download
```

This command downloads the `codeql/cpp-queries` CodeQL query pack, runs the analysis, and generates a file in the SARIF version 2.1.0 format that is supported by all versions of GitHub. This file can be uploaded to GitHub by executing `codeql github upload-results` or the code scanning API. For more information, see "Uploading CodeQL analysis results to GitHub" or "Code Scanning".

CodeQL query suites are `.qls` files that use directives to select queries to run based on certain metadata properties. The standard CodeQL packs have metadata that specify the location of the query suites used by code scanning, so the CodeQL CLI knows where to find these suite files automatically, and you don't have to specify the full path on the command line. For more information, see "Creating CodeQL query suites."

For information about creating custom query suites, see "Creating CodeQL query suites."

Results

You can save analysis results in a number of different formats, including SARIF and CSV.

The SARIF format is designed to represent the output of a broad range of static analysis tools. For more information, see "CodeQL CLI SARIF output."

For more information about what the results look like in CSV format, see "CodeQL CLI CSV output."

Results files can be integrated into your own code-review or debugging infrastructure. For example, SARIF file output can be used to highlight alerts in the correct location in your source code using a SARIF viewer plugin for your IDE.

Viewing log and diagnostic information

When you analyze a CodeQL database using a code scanning query suite, in addition to generating detailed information about alerts, the CLI reports diagnostic data from the database generation step and summary metrics. If you choose to generate SARIF output, the additional data is also included in the SARIF file. For repositories with few alerts, you may find this information useful for determining if there are genuinely few problems in the code, or if there were errors generating the CodeQL database. For more detailed output from `codeql database analyze`, use the `--verbose` option.

For more information about the type of diagnostic information available, see "Viewing code scanning logs".

You can choose to export and upload diagnostic information to GitHub Enterprise Cloud

even if a CodeQL analysis fails. For more information, see "Uploading CodeQL analysis results to GitHub."

Next steps

- To learn how to upload your CodeQL analysis results to GitHub Enterprise Cloud, see "Uploading CodeQL analysis results to GitHub."