Publishing and using CodeQL packs

In this article

You can publish your own CodeQL packs and use packs published by other people.

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 Container registry. To use this beta functionality, install the latest version of the CodeQL CLI bundle from: https://github.com/github/codeql-action/releases.

Configuring the `qlpack.yml` file before publishing

Note: This article describes the features available with the CodeQL CLI 2.13.5 bundle included in the initial release of GitHub Enterprise Server 3.10.

If your site administrator has updated your CodeQL CLI version to a newer release, please see the GitHub Enterprise Cloud version of this article for information on the latest features.

You can check and modify the configuration details of your CodeQL pack prior to publishing. Open the `qlpack.yml` file in your preferred text editor.

```
library: # set to true if the pack is a library. Set to false or omit for a query pack
name: <scope>/<pack>
version: <x.x.x>
description: <Description to publish with the package>
default-suite: # optional, one or more queries in the pack to run by default
```

```
    - query: <relative-path>/query-file>.ql
default-suite-file: default-queries.qls # optional, a pointer to a query-suite in
this pack
license: # optional, the license under which the pack is published
dependencies: # map from CodeQL pack name to version range
```

- `name:` must follow the `<scope>/<pack>` format, where `<scope>` is the GitHub organization that you will publish to and is the name for the pack.
- A maximum of one of `default-suite` or `default-suite-file` is allowed. These are two different ways to define a default query suite to be run, the first by specifying queries directly in the qlpack.yml file and the second by specifying a query suite in the pack.

Running `codeql pack publish`

When you are ready to publish a pack to the GitHub Container registry, you can run the following command in the root of the pack directory:

```
codeql pack publish
```

The published package will be displayed in the packages section of GitHub organization specified by the scope in the `qlpack.yml` file.

Running `codeql pack download <scope>/<pack>`

To run a pack that someone else has created, you must first download it by running the following command:

```
codeql pack download <scope>/<pack>@x.x.x
```

- `<scope>`: the name of the GitHub organization that you will download from.
- `<pack>`: the name for the pack that you want to download.
- `@x.x.x`: an optional version number. If omitted, the latest version will be downloaded.

This command accepts arguments for multiple packs.

If you write scripts that specify a particular version number of a query pack to download, keep in mind that when you update your version of CodeQL to a newer one, you may also need to switch to a newer version of the query pack. Newer versions of CodeQL *may* provide degraded performance when used with query packs that have been pinned to a very old version. For more information, see "About CodeQL pack compatibility."

Using a CodeQL pack to analyze a CodeQL database

To analyze a CodeQL database with a CodeQL pack, run the following command:

```
codeql database analyze <database> <scope>/<pack>@x.x.x:<path>
```

- `<database>`: the CodeQL database to be analyzed.
- `<scope>`: the name of the GitHub organization that the pack is published to.
- `<pack>`: the name for the pack that you are using.
- `@x.x.x`: an optional version number. If omitted, the latest version will be used.

- `:<path>`: an optional path to a query, directory, or query suite. If omitted, the pack's default query suite will be used.

The `analyze` command will run the default suite of any specified CodeQL packs. You can specify multiple CodeQL packs to be used for analyzing a CodeQL database. For example:

```
codeql <database> analyze <scope>/<pack> <scope>/<other-pack>
```

Note: The `codeql pack download` command stores the pack it downloads in an internal location that is not intended for local modification. Unexpected (and hard to troubleshoot) behavior may result if the pack is modified after downloading. For more information about customizing packs, see "Creating and working with CodeQL packs."

About CodeQL pack compatibility

When a query pack is published, it includes pre-compiled representations of all the queries in it. These pre-compiled queries are generally much faster to execute than it is to compile the QL source from scratch during the analysis. However, the pre-compiled queries also depend on certain internals of the QL evaluator, so if the version of CodeQL that performs the analysis is too different from the version that ran `codeql pack publish`, it may be necessary to compile the queries from source instead during analysis. The recompilation happens automatically and will not affect the *results* of the analysis, but it can make the analysis significantly slower.

It can generally be assumed that if a pack is published with one release of CodeQL, the precompiled queries in it can be used directly by *later* releases of CodeQL, as long as there is no more than 6 months between the release dates. We will make reasonable efforts to keep new releases compatible for longer than that, but make no promises.

It can also be assumed that a pack published by the *latest* public release of CodeQL will be useable by the version of CodeQL that is used by code scanning and GitHub Actions, even though that is often a slightly older release.

As an exception to the above, packs published with versions of CodeQL *earlier than 2.12.0* are not compatible with any earlier or later versions. These old versions did not write pre-compiled queries in a format that supported compatibility between releases. Packs published by these versions can still be *used* by newer versions, but the analysis will be slower because the queries have to be recompiled first.

As a user of a published query pack, you can check that the CodeQL makes use of the precompiled queries in it by inspecting the terminal output from an analysis runs that uses the query pack. If it contains lines looking like the following, then the precompiled queries were used successfully:

```
[42/108] Loaded /long/path/to/query/Filename.qlx.
```

However, if they instead look like the following, then usage of the precompiled queries failed:

```
Compiling query plan for /long/path/to/query/Filename.ql.
[42/108 comp 25s] Compiled /long/path/to/query/Filename.ql.
```

The results of the analysis will still be good in this case, but to get optimal performance you may need to upgrade to a newer version of the CodeQL CLI and/or of the query pack.

If you publish query packs on the Container registry on GitHub.com for others to use, we

recommend that you use a recent release of CodeQL to run `codeql pack publish`, and that you publish a fresh version of your pack with an updated CodeQL version before the version you used turns 6 months old. That way you can ensure that users of your pack who keep *their* CodeQL up to date will benefit from the pre-compiled queries in your pack.

If you publish query packs with the intention of using them on a GitHub Enterprise Server installation that uses its bundled CodeQL binaries, use the same CodeQL version to run `codeql pack publish`. Newer versions might produce pre-compiled queries that the one in GitHub Enterprise Server may not recognize. Your GitHub Enterprise Server administrator may choose to upgrade to a newer version of CodeQL periodically. If so, follow their lead.

Working with CodeQL packs on GitHub Enterprise Server

By default, the CodeQL CLI expects to download CodeQL packs from and publish packs to the Container registry on GitHub.com. However, you can also work with CodeQL packs in a Container registry on GitHub Enterprise Server by creating a `qlconfig.yml` file to tell the CLI which Container registry to use for each pack.

Create a `~/.codeql/qlconfig.yml` file using your preferred text editor, and add entries to specify which registry to use for one or more package name patterns. For example, the following `qlconfig.yml` file associates all packs with the Container registry for the GitHub Enterprise Server at `GHE_HOSTNAME`, except packs matching `codeql/\*`, which are associated with the Container registry on GitHub.com:

```yaml
registries:
- packages:
  - 'codeql/*'
  - 'other-org/*'
  url: https://ghcr.io/v2/
- packages: '*'
  url: https://containers.GHE_HOSTNAME/v2/
```

The CodeQL CLI will determine which registry to use for a given package name by finding the first item in the `registries` list with a `packages` property that matches that package name. This means that you'll generally want to define the most specific package name patterns first. The `packages` property may be a single package name, a glob pattern, or a YAML list of package names and glob patterns.

The `registries` list can also be placed inside of a `codeql-workspace.yml` file. Doing so will allow you to define the registries to be used within a specific workspace, so that it can be shared amongst other CodeQL users of the workspace. The `registries` list in the `codeql-workspace.yml` will be merged with and take precedence over the list in the global `qlconfig.yml`. For more information about `codeql-workspace.yml`, see "About CodeQL workspaces."

You can now use `codeql pack publish`, `codeql pack download`, and `codeql database analyze` to manage packs on GitHub Enterprise Server.

Authenticating to GitHub Container registries

You can publish packs and download private packs by authenticating to the appropriate GitHub Container registry.

You can authenticate to the Container registry on GitHub.com in two ways:

1. Pass the `--github-auth-stdin` option to the CodeQL CLI, then supply a GitHub Apps

token or personal access token via standard input.

2. Set the `GITHUB_TOKEN` environment variable to a GitHub Apps token or personal access token.

Similarly, you can authenticate to a GitHub Enterprise Server Container registry, or authenticate to multiple registries simultaneously (for example, to download or run private packs from multiple registries) in two ways:

1. Pass the `--registries-auth-stdin` option to the CodeQL CLI, then supply a registry authentication string via standard input.

2. Set the `CODEQL_REGISTRIES_AUTH` environment variable to a registry authentication string.

A registry authentication string is a comma-separated list of `<registry-url>=<token>` pairs, where `registry-url` is a Container registry URL, such as `https://containers.GHE_HOSTNAME/v2/`, and `token` is a GitHub Apps token or personal access token for that GitHub Container registry. This ensures that each token is only passed to the Container registry you specify. For instance, the following registry authentication string specifies that the CodeQL CLI should authenticate to the Container registry on GitHub.com using the token `<token1>` and to the Container registry for the GHES instance at `GHE_HOSTNAME` using the token `<token2>`:

```
https://ghcr.io/v2/=<token1>,https://containers.GHE_HOSTNAME/v2/=<token2>
```

About `qlpack.yml` files

When executing query-related commands, CodeQL first looks in siblings of the installation directory (and their subdirectories) for `qlpack.yml` files. Then it checks the package cache for CodeQL packs which have been downloaded. This means that when you are developing queries locally, the local packages in the installation directory override packages of the same name in the package cache, so that you can test your local changes.

The metadata in each `qlpack.yml` file tells CodeQL how to compile any queries in the pack, what libraries the pack depends on, and where to find query suite definitions.

The contents of the CodeQL pack (queries or libraries used in CodeQL analysis) is included in the same directory as `qlpack.yml`, or its subdirectories.

The directory containing the `qlpack.yml` file serves as the root directory for the content of the CodeQL pack. That is, for all `.ql` and `.qll` files in the pack, CodeQL will resolve all import statements relative to the directory containing the `qlpack.yml` file at the pack's root.

`qlpack.yml` properties

The following properties are supported in `qlpack.yml` files.

`name`

- Required by all packs.
- Defines the scope of the pack, where the CodeQL pack is published, and the name of the pack defined using alphanumeric characters and hyphens. It must be unique as CodeQL cannot differentiate between CodeQL packs with identical names. Use the

pack name to specify queries to run using `database analyze` and to define dependencies between CodeQL packs (see examples below). For example:

```
name: octo-org/security-queries
```

`version`

- Required by all packs that are published.
- Defines a semantic version for this CodeQL pack that must adhere to the SemVer v2.0.0 specification. For example:

```
version: 0.0.0
```

`dependencies`

- Required by query and library packs that define CodeQL package dependencies on other packs. Model packs cannot define any dependencies and use `extensionTargets` instead.
- Defines a map from pack references to the semantic version range that is compatible with this pack. Supported for CodeQL CLI versions v2.6.0 and later. For example:

```
dependencies:
  codeql/cpp-all: ^0.0.2
```

`defaultSuiteFile`

- Required by packs that export a set of default queries to run.
- Defines the path to a query suite file relative to the package root, containing all of the queries that are run by default when this pack is passed to the `codeql database analyze` command. Supported from CLI version v2.6.0 and onwards. Only one of `defaultSuiteFile` or `defaultSuite` can be defined. For example:

```
defaultSuiteFile: cpp-code-scanning.qls
```

`defaultSuite`

- Required by packs that export a set of default queries to run.
- Defines an inlined query suite containing all of the queries that are run by default when this pack is passed to the `codeql database analyze` command. Supported from CLI version v2.6.0 and onwards. Only one of `defaultSuiteFile` or `defaultSuite` can be defined. For example:

```
defaultSuite:
  queries: .
  exclude:
    precision: medium
```

`groups`

- Optional.
- Defines logical groupings of packs in a CodeQL workspace. Using groups is a way to

apply pack operations to subsets of packs in a workspace. For example, the following pack is defined to be a part of the `java` and the `experimental` groups:

```
groups:
  - java
  - experimental
```

Running `codeql pack publish --groups java,-experimental` will publish all of the packs in the `java` group, *except* the `experimental` packs. You can run the `codeql pack ls --groups [-]<group>[,[-]<group>...]` command to list the packs in a workspace that match the specified set of groups.

A CodeQL pack in the given workspace is included in the list if:

- It is in at least one of the groups listed without a minus sign (this condition is automatically satisfied if there are no groups listed without a minus sign), and
- It is not in any group listed with a minus sign.

`library`

- Required by library packs.
- Defines a boolean value that indicates whether or not this pack is a library pack. Library packs do not contain queries and are not compiled. Query packs can ignore this field or explicitly set it to `false`. For example:

```
library: true
```

`suites`

- Optional for packs that define query suites.
- Defines the path to a directory in the pack that contains the query suites you want to make known to the CodeQL CLI, defined relative to the pack directory. CodeQL pack users can run "well-known" suites stored in this directory by specifying the pack name, without providing their full path. This is not supported for CodeQL packs downloaded from the Container registry. For more information about query suites, see "Creating CodeQL query suites." For example:

```
suites: octo-org-query-suites
```

`tests`

- Optional for packs containing CodeQL tests. Ignored for packs without tests.
- Defines the path to a directory within the pack that contains tests, defined relative to the pack directory. Use `.` to specify the whole pack. Any queries in this directory are run as tests when `test run` is run with the `--strict-test-discovery` option. These queries are ignored by query suite definitions that use `queries` or `qlpack` instructions to ask for all queries in a particular pack. If this property is missing, then `.` is assumed. For example:

```
tests: .
```

`extractor`

- Required by all packs containing CodeQL tests.

- Defines the CodeQL language extractor to use when running the CodeQL tests in the pack. For more information about testing queries, see "Testing custom queries." For example:

```
extractor: javascript
```

`authors`

- Optional.
- Defines metadata that will be displayed on the packaging search page in the packages section of the account that the CodeQL pack is published to. For example:

```
authors: author1@github.com,author2@github.com
```

`license`

- Optional.
- Defines metadata that will be displayed on the packaging search page in the packages section of the account that the CodeQL pack is published to. For a list of allowed licenses, see SPDX License List in the SPDX Specification. For example:

```
license: MIT
```

`description`

- Optional.
- Defines metadata that will be displayed on the packaging search page in the packages section of the account that the CodeQL pack is published to. For example:

```
description: Human-readable description of the contents of the CodeQL pack.
```

`libraryPathDependencies`

- Optional, deprecated. Use the `dependencies` property instead.
- Previously used to define the names of any CodeQL packs that this CodeQL pack depends on, as an array. This gives the pack access to any libraries, database schema, and query suites defined in the dependency. For example:

```
libraryPathDependencies: codeql/javascript-all
```

`dbscheme`

- Required by core language packs only.
- Defines the path to the database schema for all libraries and queries written for this CodeQL language (see example below). For example:

```
dbscheme: semmlecode.python.dbscheme
```

`upgrades`

- Required by core language packs only.
- Defines the path to a directory within the pack that contains database upgrade scripts, defined relative to the pack directory. Database upgrades are used internally to ensure that a database created with a different version of the CodeQL CLI is compatible with the current version of the CLI. For example:

  ```
  upgrades: .
  ```

`warnOnImplicitThis`

- Optional. Defaults to `false` if the `warnOnImplicitThis` property is not defined.
- Defines a boolean that specifies whether or not the compiler should emit warnings about member predicate calls with implicit `this` call receivers, that is, without an explicit receiver. Supported from CodeQL CLI version 2.13.2 and onwards. For example:

  ```
  warnOnImplicitThis: true
  ```

About `codeql-pack.lock.yml` files

`codeql-pack.lock.yml` files store the versions of the resolved transitive dependencies of a CodeQL pack. This file is created by the `codeql pack install` command if it does not already exist and should be added to your version control system. The `dependencies` section of the `qlpack.yml` file contains version ranges that are compatible with the pack. The `codeql-pack.lock.yml` file locks the versions to precise dependencies. This ensures that running `codeql pack install` on this the pack will always retrieve the same versions of dependencies even if newer compatible versions exist.

For example, if a `qlpack.yml` file contains the following dependencies:

```
dependencies:
  codeql/cpp-all: ^0.1.2
  my-user/my-lib: ^0.2.3
  other-dependency/from-source: "*"
```

The `codeql-pack.lock.yml` file will contain something like the following:

```
dependencies:
  codeql/cpp-all:
    version: 0.1.4
  my-user/my-lib:
    version: 0.2.4
  my-user/transitive-dependency:
    version: 1.2.4
```

The `codeql/cpp-all` dependency is locked to version 0.1.4. The `my-user/my-lib` dependency is locked to version 0.2.4. The `my-user/transitive-dependency`, which is a transitive dependency and is not specified in the `qlpack.yml` file, is locked to version 1.2.4. The `other-dependency/from-source` is absent from the lock file since it is resolved from source. This dependency must be available in the same CodeQL workspace as the pack. For more information about CodeQL workspaces and resolving dependencies from source, see "About CodeQL workspaces."

In most cases, the `codeql-pack.lock.yml` file is only relevant for query packs since library packs are non-executable and usually do not need their transitive dependencies

to be fixed. The exception to this is for library packs that contain tests. In this case, the `codeql-pack.lock.yml` file is used to ensure that the tests are always run with the same versions of dependencies to avoid spurious failures when there are mismatched dependencies.

Examples of custom CodeQL packs

When you write custom queries or tests, you should save them in custom CodeQL packs. For simplicity, try to organize each pack logically. For more information, see "Creating and working with CodeQL packs." Save files for queries and tests in separate packs and, where possible, organize custom packs into specific folders for each target language. This is particularly useful if you intend to publish your CodeQL packs so they can be shared with others or used in code scanning. For more information, see "About code scanning with CodeQL."

CodeQL packs for custom libraries

A custom CodeQL pack containing custom C++ libraries, with no queries or tests, may have a `qlpack.yml` file containing:

```
name: my-github-user/my-custom-libraries
version: 1.2.3
library: true
dependencies:
  codeql/cpp-all: ^0.1.2
```

where `codeql/cpp-all` is the name of the CodeQL pack for C/C++ analysis included in the CodeQL repository. The version range `^0.1.2` indicates that this pack is compatible with all versions of `codeql/cpp-all` that are greater than or equal to `0.1.2` and less than `0.2.0`. Any CodeQL library file (a file with a `.qll` extension) defined in this pack will be available to queries defined in any query pack that includes this pack in its dependencies block.

The `library` property indicates that this pack is a library pack and does not contain any queries.

CodeQL packs for custom queries

A custom CodeQL pack containing custom C++ queries and libraries may have a `qlpack.yml` file containing:

```
name: my-github-user/my-custom-queries
version: 1.2.3
dependencies:
  codeql/cpp-all: ^0.1.2
  my-github-user/my-custom-libraries: ^1.2.3
suites: my-custom-suites
```

where `codeql/cpp-all` is the name of the CodeQL pack for C/C++ analysis included in the CodeQL repository. The version range `^0.1.2` indicates that this pack is compatible with all versions of `codeql/cpp-all` that are greater than or equal to `0.1.2` and less than `0.2.0`. `my-github-user/my-custom-libraries` is the name of a CodeQL pack containing custom CodeQL libraries for C++. Any CodeQL library file (a file with a `.qll` extension) defined in this pack will be available to queries in the `my-github-user/my-custom-queries` pack.

The `suites` property indicates a directory where "well-known" query suites can be found. These suites can be used on the command line by referring to their name only,

rather than their full path. For more information about query suites, see "Creating CodeQL query suites."

CodeQL packs for custom tests

For custom CodeQL packs containing test files, you also need to include an `extractor` property so that the `test run` command knows how to create test databases. You may also wish to specify the `tests` property.

The following `qlpack.yml` file states that `my-github-user/my-query-tests` depends on `my-github-user/my-custom-queries` at a version greater than or equal to 1.2.3 and less than 2.0.0. It also declares that the CLI should use the Java `extractor` when creating test databases. The `tests: .` line declares that all `.ql` files in the pack should be run as tests when `codeql test run` is run with the `--strict-test-discovery` option. Typically, test packs do not contain a `version` property. This prevents you from accidentally publishing them.

```yaml
name: my-github-user/my-query-tests
dependencies:
  my-github-user/my-custom-queries: ^1.2.3
extractor: java
tests: .
```

For more information about running tests, see "Testing custom queries."

Examples of CodeQL packs in the CodeQL repository

Each of the languages in the CodeQL repository has four main CodeQL packs:

- Core library pack for the language, with the database schema used by the language, and CodeQL libraries, and queries at `<language>/ql/lib`
- Core query pack for the language that includes the default queries for the language, along with their query suites at `<language>/ql/src`
- Tests for the core language libraries and queries at `<language>/ql/test`
- Example queries for the language at `<language>/ql/examples`

Core library pack

Here is an example `qlpack.yml` file for the C/C++ analysis libraries core language pack:

```yaml
name: codeql/cpp-all
version: x.y.z-dev
dbscheme: semmlecode.cpp.dbscheme
library: true
upgrades: upgrades
```

Some extra notes on the following properties:

- `library`: Indicates that this is a library pack with no executable queries. It is only meant to be used as a dependency for other packs.
- `dbscheme` and `upgrades`: These properties are internal to the CodeQL CLI and should only be defined in the core QL pack for a language.

Core query pack

Here is an example `qlpack.yml` file for C/C++ analysis queries core query pack:

```
name: codeql/cpp-queries
version: x.y.z-dev
dependencies:
    codeql/cpp-all: "*"
    codeql/suite-helpers: "*"
suites: codeql-suites
defaultSuiteFile: codeql-suites/cpp-code-scanning.qls
```

Some extra notes on the following properties:

- `dependencies`: This query pack depends on `codeql/cpp-all` and `codeql/suite-helpers`. Since these dependencies are resolved from source, it does not matter what version of the CodeQL pack they are compatible with. For more information about resolving dependencies from source, see "Source Dependencies."
- `suites`: Indicates the directory containing "well-known" query suites.
- `defaultSuiteFile`: The name of the default query suite file that is used when no query suite is specified.

Tests for the core CodeQL pack

Here is an example `qlpack.yml` file for C/C++ analysis tests core test pack:

```
name: codeql/cpp-tests
dependencies:
  codeql/cpp-all: "*"
  codeql/cpp-queries: "*"
extractor: cpp
tests: .
```

Some extra notes on the following properties:

- `dependencies`: This pack depends on the core CodeQL query and library packs for C++.
- `extractor`: This specifies that all the tests will use the same C++ extractor to create the database for the tests.
- `tests`: This specifies the location of the tests. In this case, the tests are in the root folder (and all sub-folders) of the pack.
- `version`: There is no `version` property for the tests pack. This prevents test packs from accidentally being published.