Customizing your advanced setup for code scanning

In this article

About code scanning configuration
Editing a code scanning workflow
Configuring frequency
Specifying an operating system
Specifying the location for CodeQL databases
Changing the languages that are analyzed
Analyzing Python dependencies
Defining the alert severities that give a check failure for a pull request
Configuring a category for the analysis
Extending CodeQL coverage with CodeQL model packs
Running additional queries
Using a custom configuration file
Specifying configuration details using the config input
Configuring code scanning for compiled languages
Uploading code scanning data to GitHub

You can customize how your advanced setup scans the code in your project for vulnerabilities and errors.

> **Who can use this feature**
> People with write permissions to a repository can customize code scanning for the repository.

Code scanning is available for all public repositories on GitHub.com. To use code scanning in a private repository owned by an organization, you must have a license for GitHub Advanced Security. For more information, see "About GitHub Advanced Security."

About code scanning configuration

You can run code scanning on GitHub Enterprise Cloud, using GitHub Actions, or from your continuous integration (CI) system. For more information, see "Learn GitHub Actions" or "Using code scanning with your existing CI system."

With advanced setup for code scanning, you can customize a code scanning workflow for granular control over your configuration. For more information, see "Configuring advanced setup for code scanning."

CodeQL analysis is just one type of code scanning you can do in GitHub. GitHub Marketplace contains other code scanning workflows you can use. You can find a selection of these on the "Get started with code scanning" page, which you can access from the **Security** tab. The specific examples given in this article relate to the CodeQL analysis workflow file.

Editing a code scanning workflow

GitHub saves workflow files in the *.github/workflows* directory of your repository. You can find a workflow you have added by searching for its file name. For example, by default, the workflow file for CodeQL code scanning is called *codeql-analysis.yml*.

1. In your repository, browse to the workflow file you want to edit.

2. In the upper right corner of the file view, to open the workflow editor, click ✎.

3. After you have edited the file, click **Start commit** and complete the "Commit changes" form. You can choose to commit directly to the current branch, or create a new branch and start a pull request.

For more information about editing workflow files, see "Learn GitHub Actions."

Configuring frequency

You can configure the CodeQL analysis workflow to scan code on a schedule or when specific events occur in a repository.

Scanning code when someone pushes a change, and whenever a pull request is created, prevents developers from introducing new vulnerabilities and errors into the code. Scanning code on a schedule informs you about the latest vulnerabilities and errors that GitHub, security researchers, and the community discover, even when developers aren't actively maintaining the repository.

Scanning on push

By default, the CodeQL analysis workflow uses the `on.push` event to trigger a code scan on every push to the default branch of the repository and any protected branches. For code scanning to be triggered on a specified branch, the workflow must exist in that branch. For more information, see "Workflow syntax for GitHub Actions."

If you scan on push, then the results appear in the **Security** tab for your repository. For more information, see "Managing code scanning alerts for your repository."

Additionally, when an `on:push` scan returns results that can be mapped to an open pull request, these alerts will automatically appear on the pull request in the same places as other pull request alerts. The alerts are identified by comparing the existing analysis of the head of the branch to the analysis for the target branch. For more information on code scanning alerts in pull requests, see "Triaging code scanning alerts in pull requests."

Scanning pull requests

The default CodeQL analysis workflow uses the `pull_request` event to trigger a code scan on pull requests targeted against the default branch. If a pull request is from a private fork, the `pull_request` event will only be triggered if you've selected the "Run workflows from fork pull requests" option in the repository settings. For more information, see "Managing GitHub Actions settings for a repository."

For more information about the `pull_request` event, see "Events that trigger workflows."

If you scan pull requests, then the results appear as alerts in a pull request check. For more information, see "Triaging code scanning alerts in pull requests."

Using the `pull_request` trigger, configured to scan the pull request's merge commit rather than the head commit, will produce more efficient and accurate results than scanning the head of the branch on each push. However, if you use a CI/CD system that cannot be configured to trigger on pull requests, you can still use the `on:push` trigger and code scanning will map the results to open pull requests on the branch and add the alerts as annotations on the pull request. For more information, see "Scanning on push."

> **Note:** If your repository is configured with a merge queue, you need to include the `merge_group` event as an additional trigger for code scanning. This will ensure that pull requests are also scanned when they are added to a merge queue. For more information, see "Managing a merge queue."

Avoiding unnecessary scans of pull requests

You might want to avoid a code scan being triggered on specific pull requests targeted against the default branch, irrespective of which files have been changed. You can configure this by specifying `on:pull_request:paths-ignore` or `on:pull_request:paths` in the code scanning workflow. For example, if the only changes in a pull request are to files with the file extensions `.md` or `.txt` you can use the following `paths-ignore` array.

YAML

```yaml
on:
  push:
    branches: [main, protected]
  pull_request:
    branches: [main]
    paths-ignore:
      - '**/*.md'
      - '**/*.txt'
```

> **Notes**
>
> - `on:pull_request:paths-ignore` and `on:pull_request:paths` set conditions that determine whether the actions in the workflow will run on a pull request. They don't determine what files will be analyzed when the actions *are* run. When a pull request contains any files that are not matched by `on:pull_request:paths-ignore` or `on:pull_request:paths`, the workflow runs the actions and scans all of the files changed in the pull request, including those matched by `on:pull_request:paths-ignore` or `on:pull_request:paths`, unless the files have been excluded. For information on how to exclude files from analysis, see "Specifying directories to scan."

For more information about using `on:pull_request:paths-ignore` and `on:pull_request:paths` to determine when a workflow will run for a pull request, see "Workflow syntax for GitHub Actions."

Scanning on a schedule

If you use the default CodeQL analysis workflow, the workflow will scan the code in your repository once a week, in addition to the scans triggered by events. To adjust this schedule, edit the `cron` value in the workflow. For more information, see "Workflow syntax for GitHub Actions."

> **Note**: GitHub only runs scheduled jobs that are in workflows on the default branch. Changing the schedule in a workflow on any other branch has no effect until you merge the branch into the default branch.

Example

The following example shows a CodeQL analysis workflow for a particular repository that has a default branch called `main` and one protected branch called `protected`.

YAML

```yaml
on:
  push:
    branches: [main, protected]
  pull_request:
    branches: [main]
  schedule:
    - cron: '20 14 * * 1'
```

This workflow scans:

- Every push to the default branch and the protected branch
- Every pull request to the default branch
- The default branch every Monday at 14:20 UTC

Specifying an operating system

Notes:

- Code scanning of Swift code uses macOS runners by default. GitHub-hosted macOS runners are more expensive than Linux and Windows runners, so you should consider only scanning the build step. For more information about configuring code scanning for Swift, see "CodeQL code scanning for compiled languages." For more information about pricing for GitHub-hosted runners, see "About billing for GitHub Actions."
- Code scanning of Swift code is not supported for runners that are part of an Actions Runner Controller (ARC), because ARC runners only use Linux and Swift requires macOS runners. However, you can have a mixture of both ARC runners and self-hosted macOS runners. For more information, see "About Actions Runner Controller."

If your code requires a specific operating system to compile, you can configure the operating system in your CodeQL analysis workflow. Edit the value of `jobs.analyze.runs-on` to specify the operating system for the machine that runs your code scanning actions.

YAML

```yaml
jobs:
  analyze:
    name: Analyze
    runs-on: [ubuntu-latest]
```

If you choose to use a self-hosted runner for code scanning, you can specify an operating system by using an appropriate label as the second element in a two-element array, after `self-hosted`.

YAML

```yaml
jobs:
  analyze:
    name: Analyze
    runs-on: [self-hosted, ubuntu-latest]
```

CodeQL code scanning supports the latest versions of Ubuntu, Windows, and macOS.

Typical values for this setting are therefore: `ubuntu-latest`, `windows-latest`, and `macos-latest`. For more information, see "Choosing the runner for a job" and "Using labels with self-hosted runners."

If you use a self-hosted runner, you must ensure that Git is in the PATH variable. For more information, see "About self-hosted runners" and "Adding self-hosted runners."

For recommended specifications (RAM, CPU cores, and disk) for running CodeQL analysis on self-hosted machines, see "Recommended hardware resources for running CodeQL."

Specifying the location for CodeQL databases

In general, you do not need to worry about where the CodeQL analysis workflow places CodeQL databases since later steps will automatically find databases created by previous steps. However, if you are writing a custom workflow step that requires the CodeQL database to be in a specific disk location, for example to upload the database as a workflow artifact, you can specify that location using the `db-location` parameter under the `init` action.

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    db-location: '${{ github.runner_temp }}/my_location'
```

The CodeQL analysis workflow will expect the path provided in `db-location` to be writable, and either not exist, or be an empty directory. When using this parameter in a job running on a self-hosted runner or using a Docker container, it's the responsibility of the user to ensure that the chosen directory is cleared between runs, or that the databases are removed once they are no longer needed. This is not necessary for jobs running on GitHub-hosted runners, which obtain a fresh instance and a clean filesystem each time they run. For more information, see "Using GitHub-hosted runners."

If this parameter is not used, the CodeQL analysis workflow will create databases in a temporary location of its own choice. Currently the default value is `${{ github.runner_temp }}/codeql_databases`.

Changing the languages that are analyzed

CodeQL code scanning automatically detects code written in the supported languages.

- C/C++
- C#
- Go
- Java/Kotlin
- JavaScript/TypeScript
- Python
- Ruby
- Swift

Notes:

- CodeQL analysis for Swift is currently in beta. During the beta, analysis of Swift will be less comprehensive than CodeQL analysis of other languages. Additionally, Swift 5.8 is not yet supported.

- CodeQL analysis for Kotlin is currently in beta. During the beta, analysis of Kotlin will be less comprehensive than CodeQL analysis of other languages.

- Use `java-kotlin` to analyze code written in Java, Kotlin or both.
- Use `javascript-typescript` to analyze code written in JavaScript, TypeScript or both.

For more information, see the documentation on the CodeQL website: "Supported languages and frameworks."

CodeQL uses the following language identifiers:

Language	Identifier	Optional alternative identifiers (if any)
C/C++	`c-cpp`	`c` or `cpp`
C#	`csharp`	
Go	`go`	
Java/Kotlin	`java-kotlin`	`java` or `kotlin`
JavaScript/TypeScript	`javascript-typescript`	`javascript` or `typescript`
Python	`python`	
Ruby	`ruby`	
Swift	`swift`	

Note: If you specify one of the alternative identifiers, this is equivalent to using the standard language identifier. For example, specifying `javascript` instead of `javascript-typescript` will not exclude analysis of TypeScript code. You can do this in an advanced setup workflow with the `--paths-ignore` option. For more information, see "Customizing your advanced setup for code scanning."

The default CodeQL analysis workflow file contains a matrix called `language` which lists the languages in your repository that are analyzed. CodeQL automatically populates this matrix when you add code scanning to a repository. Using the `language` matrix optimizes CodeQL to run each analysis in parallel. We recommend that all workflows adopt this configuration due to the performance benefits of parallelizing builds. For more information about matrices, see "Using a matrix for your jobs."

If your repository contains code in more than one of the supported languages, you can choose which languages you want to analyze. There are several reasons you might want to prevent a language being analyzed. For example, the project might have dependencies in a different language to the main body of your code, and you might prefer not to see alerts for those dependencies.

If your workflow uses the `language` matrix then CodeQL is hardcoded to analyze only the languages in the matrix. To change the languages you want to analyze, edit the value of the matrix variable. You can remove a language to prevent it being analyzed or you can add a language that was not present in the repository when code scanning was configured. For example, if the repository initially only contained JavaScript when code scanning was configured, and you later added Python code, you will need to add `python` to the matrix.

YAML

```yaml
jobs:
  analyze:
    name: Analyze
    ...
    strategy:
      fail-fast: false
      matrix:
        language: ['javascript-typescript', 'python']
```

If your workflow does not contain a matrix called `language`, then CodeQL is configured to run analysis sequentially. If you don't specify languages in the workflow, CodeQL automatically detects, and attempts to analyze, any supported languages in the repository. If you want to choose which languages to analyze, without using a matrix, you can use the `languages` parameter under the `init` action.

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    languages: c-cpp, csharp, python
```

Analyzing Python dependencies

Notes:

- As of July 12, 2023, automatic dependency installation is disabled by default for new users of CodeQL for Python, with new users defined as those who have no prior Python projects set up for code scanning with CodeQL via advanced setup.
- Existing code scanning users that have already set up CodeQL to scan at least one Python project will not see any changes in behavior, even to newly configured repositories. However, for improved scan times, we encourage users to disable dependency installation by setting `setup-python-dependencies: false` in the "Initialize CodeQL" step of the workflow.
- Automatic installation of dependencies will be deprecated for all users by the end of 2023.

For GitHub-hosted runners that use Linux only, the CodeQL analysis workflow will try to auto-install Python dependencies to give more results for the CodeQL analysis. You can control this behavior by specifying the `setup-python-dependencies` parameter for the action called by the "Initialize CodeQL" step. By default, this parameter is set to `true`:

- If the repository contains code written in Python, the "Initialize CodeQL" step installs the necessary dependencies on the GitHub-hosted runner. If the auto-install succeeds, the action also sets the environment variable `CODEQL_PYTHON` to the Python executable file that includes the dependencies.

- If the repository doesn't have any Python dependencies, or the dependencies are specified in an unexpected way, you'll get a warning and the action will continue with the remaining jobs. The action can run successfully even when there are problems interpreting dependencies, but the results may be incomplete.

Alternatively, you can install Python dependencies manually on any operating system. You will need to add `setup-python-dependencies` and set it to `false`, as well as set `CODEQL_PYTHON` to the Python executable that includes the dependencies, as shown in this workflow extract:

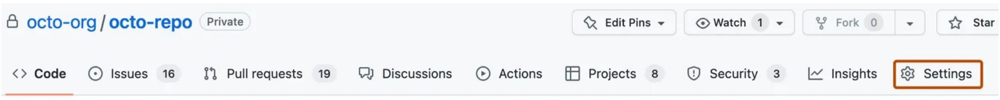
YAML

```yaml
jobs:
  CodeQL-Build:
    runs-on: ubuntu-latest
    permissions:
      security-events: write
      actions: read
    steps:
      - name: Checkout repository
        uses: actions/checkout@v4
      - name: Set up Python
        uses: actions/setup-python@v4
        with:
          python-version: '3.x'
      - name: Install dependencies
        run: |
          python -m pip install --upgrade pip
          if [ -f requirements.txt ];
          then pip install -r requirements.txt;
          fi
          # Set the `CODEQL-PYTHON` environment variable to the Python executable
          # that includes the dependencies
          echo "CODEQL_PYTHON=$(which python)" >> $GITHUB_ENV
      - name: Initialize CodeQL
        uses: github/codeql-action/init@v2
        with:
          languages: python
          # Override the default behavior so that the action doesn't attempt
          # to auto-install Python dependencies
          setup-python-dependencies: false
```

Defining the alert severities that give a check failure for a pull request

By default, only alerts with the severity level of `Error` or security severity level of `Critical` or `High` will cause a pull request check failure, and a check will still succeed with alerts of lower severities. You can change the levels of alert severities and of security severities that will cause a pull request check failure in your repository settings. For more information about severity levels, see "About code scanning alerts."

1. On GitHub.com, navigate to the main page of the repository.
2. Under your repository name, click ⚙ **Settings**. If you cannot see the "Settings" tab, select the ··· dropdown menu, then click **Settings**.



3. In the "Security" section of the sidebar, click **Code security and analysis**.
4. Under "Code scanning", in the "Protection rules" section, use the drop-down menu to define which alerts should cause a check failure. Choose one level for alerts of type "Security" and one level for all other alerts.

Configuring a category for the analysis

Use `category` to distinguish between multiple analyses for the same tool and commit, but performed on different languages or different parts of the code. The category you specify in your workflow will be included in the SARIF results file.

This parameter is particularly useful if you work with monorepos and have multiple SARIF files for different components of the monorepo.

YAML

```
    - name: Perform CodeQL Analysis
      uses: github/codeql-action/analyze@v2
      with:
        # Optional. Specify a category to distinguish between multiple analyses
        # for the same tool and ref. If you don't use `category` in your
workflow,
        # GitHub will generate a default category name for you
        category: "my_category"
```

If you don't specify a `category` parameter in your workflow, GitHub Enterprise Cloud will generate a category name for you, based on the name of the workflow file triggering the action, the action name, and any matrix variables. For example:

- The `.github/workflows/codeql-analysis.yml` workflow and the `analyze` action will produce the category `.github/workflows/codeql.yml:analyze`.
- The `.github/workflows/codeql-analysis.yml` workflow, the `analyze` action, and the `{language: javascript-typescript, os: linux}` matrix variables will produce the category `.github/workflows/codeql-analysis.yml:analyze/language:javascript-typescript/os:linux`.

The `category` value will appear as the `<run>.automationDetails.id` property in SARIF v2.1.0. For more information, see "SARIF support for code scanning."

Your specified category will not overwrite the details of the `runAutomationDetails` object in the SARIF file, if included.

Extending CodeQL coverage with CodeQL model packs

If your codebase depends on a library or framework that is not recognized by the standard queries in CodeQL, you can extend the CodeQL coverage in your code scanning workflow by specifying published CodeQL model packs. For more information about creating your own model packs, see "Creating and working with CodeQL packs."

> **Note:** Model packs are currently in beta and subject to change. During the beta, model packs are supported only by Java analysis.

To add one or more published CodeQL model packs, specify them inside the `with: packs:` entry within the `uses: github/codeql-action/init@v2` section of the workflow. Within `packs` you specify one or more packages to use and, optionally, which version to download. Where you don't specify a version, the latest version is downloaded. If you want to use packages that are not publicly available, you need to set the `GITHUB_TOKEN` environment variable to a secret that has access to the packages. For more information, see "Automatic token authentication" and "Using secrets in GitHub Actions."

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    config-file: ./.github/codeql/codeql-config.yml
    queries: security-extended
    packs: my-company/my-java-queries@~7.8.9,my-repo/my-java-model-pack
```

In this example, the default queries will be run for Java, as well as the queries from a version greater than or equal to `7.8.9` and less than `7.9.0` of the query pack `my-company/my-java-queries`. The dependencies modeled in the latest version of the model pack `my-repo/my-java-model-pack` will be available to both the default queries and those in `my-company/my-java-queries`.

Running additional queries

When you use CodeQL to scan code, the CodeQL analysis engine generates a database from the code and runs queries on it. CodeQL analysis uses a default set of queries, but you can specify more queries to run, in addition to the default queries.

> You can also specify the queries you want to exclude from analysis, or include in the analysis. This requires the use of a custom configuration file. For more information, see "Using a custom configuration file" and "Excluding specific queries from analysis " below.

You can run extra queries if they are part of a CodeQL pack (beta) published to the GitHub Container registry or a QL pack stored in a repository. For more information, see "About code scanning with CodeQL."

The options available to specify the additional queries you want to run are:

- `packs` to install one or more CodeQL query packs (beta) and run the default query suite or queries for those packs.
- `queries` to specify a single *.ql* file, a directory containing multiple *.ql* files, a *.qls* query suite definition file, or any combination. For more information about query suite definitions, see "Creating CodeQL query suites."

You can use both `packs` and `queries` in the same workflow.

We don't recommend referencing query suites directly from the `github/codeql` repository, for example, `github/codeql/cpp/ql/src@main`. Such queries would have to be recompiled, and may not be compatible with the version of CodeQL currently active on GitHub Actions, which could lead to errors during analysis.

Using query packs

> **Note:** The CodeQL package management functionality, including CodeQL packs, is currently in beta and subject to change.

To add one or more CodeQL query packs (beta), add a `with: packs:` entry within the `uses: github/codeql-action/init@v2` section of the workflow. Within `packs` you specify one or more packages to use and, optionally, which version to download. Where you don't specify a version, the latest version is downloaded. If you want to use packages that are not publicly available, you need to set the `GITHUB_TOKEN` environment variable to a secret that has access to the packages. For more information, see "Automatic token authentication" and "Using secrets in GitHub Actions."

> **Note:** For workflows that generate CodeQL databases for multiple languages, you must instead specify the CodeQL query packs in a configuration file. For more information, see "Specifying CodeQL query packs" below.

In the example below, `scope` is the organization or personal account that published the package. When the workflow runs, the four CodeQL query packs are downloaded from GitHub Enterprise Cloud and the default queries or query suite for each pack run:

- The latest version of `pack1` is downloaded and all default queries are run.
- Version 1.2.3 of `pack2` is downloaded and all default queries are run.
- The latest version of `pack3` that is compatible with version 3.2.1 is downloaded and all queries are run.
- Version 4.5.6 of `pack4` is downloaded and only the queries found in `path/to/queries` are run.

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    # Comma-separated list of packs to download
    packs:
scope/pack1,scope/pack2@1.2.3,scope/pack3@~3.2.1,scope/pack4@4.5.6:path/to/queries
```

Note: If you specify a particular version of a query pack to use, beware that the version you specify may eventually become too old to be used efficiently by the default CodeQL engine used by the CodeQL action. To ensure optimal performance, if you need to specify exact query pack versions, you should consider reviewing periodically whether the pinned version of the query pack needs to be moved forward.

For more information about pack compatibility, see "Publishing and using CodeQL packs."

Downloading CodeQL packs from GitHub Enterprise Server

If your workflow uses packs that are published on a GitHub Enterprise Server installation, you need to tell your workflow where to find them. You can do this by using the `registries` input of the github/codeql-action/init@v2 action. This input accepts a list of `url`, `packages`, and `token` properties as shown below.

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    registries: |
      # URL to the container registry, usually in this format
      - url: https://containers.GHEHOSTNAME1/v2/

        # List of package glob patterns to be found at this registry
        packages:
          - my-company/*
          - my-company2/*

        # Token, which should be stored as a secret
        token: ${{ secrets.GHEHOSTNAME1_TOKEN }}

      # URL to the default container registry
      - url: https://ghcr.io/v2/
        # Packages can also be a string
        packages: "*/*"
        token: ${{ secrets.GHCR_TOKEN }}
```

The package patterns in the registries list are examined in order, so you should generally place the most specific package patterns first. The values for `token` must be a personal access token (classic) generated by the GitHub instance you are downloading from with the `read:packages` permission.

Notice the `|` after the `registries` property name. This is important since GitHub Actions inputs can only accept strings. Using the `|` converts the subsequent text to a string, which is parsed later by the github/codeql-action/init@v2 action.

Using queries in QL packs

To add one or more queries, add a `with: queries:` entry within the `uses: github/codeql-action/init@v2` section of the workflow. If the queries are in a private repository, use the `external-repository-token` parameter to specify a token that has access to checkout the private repository.

You can also specify query suites in the value of `queries`. Query suites are collections of queries, usually grouped by purpose or language.

YAML

```
- uses: github/codeql-action/init@v2
  with:
    # Comma-separated list of queries / packs / suites to run.
    # This may include paths or a built in suite, for example:
    # security-extended or security-and-quality.
    queries: security-extended
    # Optional. Provide a token to access queries stored in private repositories.
    external-repository-token: ${{ secrets.ACCESS_TOKEN }}
```

The following query suites are built into CodeQL code scanning and are available for use.

Query suite	Description
`security-extended`	Queries from the default suite, plus lower severity and precision queries
`security-and-quality`	Queries from `security-extended`, plus maintainability and reliability queries

Each of these query suites contains a different subset of the queries included in the built-in CodeQL query pack for that language. The query suites are automatically generated using the metadata for each query. For more information, see "[Metadata for CodeQL queries](#)."

You can identify which query suites a query is included in by browsing the [CodeQL query help documentation](#). For each query, any suites that it is included in are displayed at the top of the page with the query metadata. For example: [Arbitrary file write during zip extraction ("Zip Slip")](#) and [Client-side request forgery](#).

When you specify a query suite, the CodeQL analysis engine will run the default set of queries and any extra queries defined in the additional query suite. The `security-extended` and `security-and-quality` query suites for JavaScript contain experimental queries. For more information, see "[About code scanning alerts](#)."

Working with custom configuration files

If you also use a configuration file for custom settings, any additional packs or queries specified in your workflow are used instead of those specified in the configuration file. If you want to run the combined set of additional packs or queries, prefix the value of `packs` or `queries` in the workflow with the `+` symbol. For more information, see "[Using a custom configuration file](#)."

In the following example, the `+` symbol ensures that the specified additional packs and queries are used together with any specified in the referenced configuration file.

YAML

```
- uses: github/codeql-action/init@v2
  with:
    config-file: ./.github/codeql/codeql-config.yml
    queries: +security-and-quality,octo-org/python-qlpack/show_ifs.ql@main
    packs: +scope/pack1,scope/pack2@1.2.3,scope/pack3@4.5.6:path/to/queries
```

Using a custom configuration file

A custom configuration file is an alternative way to specify additional packs and queries to run. You can also use the file to disable the default queries, exclude or include specific queries, and to specify which directories to scan during analysis.

In the workflow file, use the `config-file` parameter of the `init` action to specify the path to the configuration file you want to use. This example loads the configuration file *./.github/codeql/codeql-config.yml*.

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    config-file: ./.github/codeql/codeql-config.yml
```

The configuration file can be located within the repository you are analyzing, or in an external repository. Using an external repository allows you to specify configuration options for multiple repositories in a single place. When you reference a configuration file located in an external repository, you can use the *OWNER/REPOSITORY/FILENAME@BRANCH* syntax. For example, *octo-org/shared/codeql-config.yml@main*.

If the configuration file is located in an external private repository, use the `external-repository-token` parameter of the `init` action to specify a token that has access to the private repository.

YAML

```yaml
- uses: github/codeql-action/init@v2
  with:
    external-repository-token: ${{ secrets.ACCESS_TOKEN }}
```

The settings in the configuration file are written in YAML format.

Specifying CodeQL query packs

> **Note:** The CodeQL package management functionality, including CodeQL packs, is currently in beta and subject to change.

You specify CodeQL query packs in an array. Note that the format is different from the format used by the workflow file.

YAML

```yaml
packs:
  # Use the latest version of 'pack1' published by 'scope'
  - scope/pack1
  # Use version 1.2.3 of 'pack2'
  - scope/pack2@1.2.3
  # Use the latest version of 'pack3' compatible with 3.2.1
  - scope/pack3@~3.2.1
  # Use pack4 and restrict it to queries found in the 'path/to/queries' directory
  - scope/pack4:path/to/queries
  # Use pack5 and restrict it to the query 'path/to/single/query.ql'
  - scope/pack5:path/to/single/query.ql
  # Use pack6 and restrict it to the query suite 'path/to/suite.qls'
  - scope/pack6:path/to/suite.qls
```

The full format for specifying a query pack is `scope/name[@version][:path]`. Both `version` and `path` are optional. `version` is semver version range. If it is missing, the latest version is used. For more information about semver ranges, see the semver docs on npm.

If you have a workflow that generates more than one CodeQL database, you can specify any CodeQL query packs to run in a custom configuration file using a nested map of packs.

YAML

```yaml
packs:
  # Use these packs for JavaScript and TypeScript analysis
  javascript:
    - scope/js-pack1
    - scope/js-pack2
  # Use these packs for Java and Kotlin analysis
  java:
    - scope/java-pack1
    - scope/java-pack2@v1.0.0
```

Specifying additional queries

You specify additional queries in a `queries` array. Each element of the array contains a `uses` parameter with a value that identifies a single query file, a directory containing query files, or a query suite definition file.

YAML

```yaml
queries:
  - uses: ./my-basic-queries/example-query.ql
  - uses: ./my-advanced-queries
  - uses: ./query-suites/my-security-queries.qls
```

Optionally, you can give each array element a name, as shown in the example configuration files below. For more information about additional queries, see "Running additional queries" above.

Disabling the default queries

If you only want to run custom queries, you can disable the default security queries by using `disable-default-queries: true`.

Excluding specific queries from analysis

You can add `exclude` and `include` filters to your custom configuration file, to specify the queries you want to exclude or include in the analysis.

This is useful if you want to exclude, for example:

- Specific queries from the default suites (`security`, `security-extended` and `security-and-quality`).
- Specific queries whose results do not interest you.
- All the queries that generate warnings and recommendations.

You can use `exclude` filters similar to those in the configuration file below to exclude queries that you want to remove from the default analysis. In the example of configuration file below, both the `js/redundant-assignment` and the `js/useless-`

assignment-to-local queries are excluded from analysis.

YAML

```yaml
query-filters:
  - exclude:
      id: js/redundant-assignment
  - exclude:
      id: js/useless-assignment-to-local
```

To find the id of a query, you can click the alert in the list of alerts in the **Security** tab. This opens the alert details page. The `Rule ID` field contains the query id. For more information about the alert details page, see "About code scanning alerts."

Tips:

- The order of the filters is important. The first filter instruction that appears after the instructions about the queries and query packs determines whether the queries are included or excluded by default.
- Subsequent instructions are executed in order and the instructions that appear later in the file take precedence over the earlier instructions.

You can find another example illustrating the use of these filters in the "Example configuration files" section.

For more information about using `exclude` and `include` filters in your custom configuration file, see "Creating CodeQL query suites." For information on the query metadata you can filter on, see "Metadata for CodeQL queries."

Specifying directories to scan

For the interpreted languages that CodeQL supports (Python, Ruby, and JavaScript/TypeScript), you can restrict code scanning to files in specific directories by adding a `paths` array to the configuration file. You can exclude the files in specific directories from analysis by adding a `paths-ignore` array.

YAML

```yaml
paths:
  - src
paths-ignore:
  - src/node_modules
  - '**/*.test.js'
```

Note:

- The `paths` and `paths-ignore` keywords, used in the context of the code scanning configuration file, should not be confused with the same keywords when used for `on.<push|pull_request>.paths` in a workflow. When they are used to modify `on.<push|pull_request>` in a workflow, they determine whether the actions will be run when someone modifies code in the specified directories. For more information, see "Workflow syntax for GitHub Actions."
- The filter pattern characters `?`, `+`, `[`, `]`, and `!` are not supported and will be matched literally.
- `**` characters can only be at the start or end of a line, or surrounded by slashes, and you can't mix `**` and other characters. For example, `foo/**`, `**/foo`, and `foo/**/bar` are all allowed syntax, but `**foo` isn't. However you can use single stars along with other characters, as shown in the example. You'll need to quote anything that contains a `*` character.

For compiled languages, if you want to limit code scanning to specific directories in your project, you must specify appropriate build steps in the workflow. The commands you need to use to exclude a directory from the build will depend on your build system. For more information, see "CodeQL code scanning for compiled languages."

You can quickly analyze small portions of a monorepo when you modify code in specific directories. You'll need to both exclude directories in your build steps and use the `paths-ignore` and `paths` keywords for `on.<push|pull_request>` in your workflow.

Example configuration files

This configuration file adds the `security-and-quality` query suite to the list of queries run by CodeQL when scanning your code. For more information about the query suites available for use, see "Running additional queries."

```yaml
name: "My CodeQL config"

queries:
  - uses: security-and-quality
```

The following configuration file disables the default queries and specifies a set of custom queries to run instead. It also configures CodeQL to scan files in the *src* directory (relative to the root), except for the *src/node_modules* directory, and except for files whose name ends in *.test.js*. Files in *src/node_modules* and files with names ending *.test.js* are therefore excluded from analysis.

```yaml
name: "My CodeQL config"

disable-default-queries: true

queries:
  - name: Use an in-repository QL pack (run queries in the my-queries directory)
    uses: ./my-queries
  - name: Use an external JavaScript QL pack (run queries from an external repo)
    uses: octo-org/javascript-qlpack@main
  - name: Use an external query (run a single query from an external QL pack)
    uses: octo-org/python-qlpack/show_ifs.ql@main
  - name: Use a query suite file (run queries from a query suite in this repo)
    uses: ./codeql-qlpacks/complex-python-qlpack/rootAndBar.qls

paths:
  - src
paths-ignore:
  - src/node_modules
  - '**/*.test.js'
```

The following configuration file only runs queries that generate alerts of severity error. The configuration first selects all the default queries, all queries in `./my-queries`, and the default suite in `codeql/java-queries`, then excludes all the queries that generate warnings or recommendations.

```yaml
queries:
  - name: Use an in-repository QL pack (run queries in the my-queries directory)
    uses: ./my-queries
packs:
  - codeql/java-queries
query-filters:
- exclude:
    problem.severity:
      - warning
      - recommendation
```

Specifying configuration details using the `config` input

If you'd prefer to specify additional configuration details in the workflow file, you can use the `config` input of the `init` command of the CodeQL action. The value of this input must be a YAML string that follows the configuration file format documented at "Using a custom configuration file" above.

Example configuration

This step in a GitHub Actions workflow file uses a `config` input to disable the default queries, add the `security-extended` query suite, and exclude queries that are tagged with `cwe-020`.

```
- uses: github/codeql-action/init@v2
  with:
    languages: ${{ matrix.language }}
    config: |
      disable-default-queries: true
      queries:
        - uses: security-extended
      query-filters:
        - exclude:
          tags: /cwe-020/
```

You can use the same approach to specify any valid configuration options in the workflow file.

Tip:

You can share one configuration across multiple repositories using GitHub Actions variables. One benefit of this approach is that you can update the configuration in a single place without editing the workflow file.

In the following example, `vars.CODEQL_CONF` is a GitHub Actions variable. Its value can be the contents of any valid configuration file. For more information, see "Variables."

```
- uses: github/codeql-action/init@v2
  with:
    languages: ${{ matrix.language }}
    config: ${{ vars.CODEQL_CONF }}
```

Configuring code scanning for compiled languages

CodeQL analyzes the C/C++, C#, Go, Java, and Swift source files in your repository that are built.

If you enable default setup, the `autobuild` action will be used to build your code, as part of your automatically configured CodeQL analysis workflow. If you enable advanced setup, the basic CodeQL analysis workflow uses `autobuild`. Alternatively, you can disable `autobuild` and instead specify explicit build commands to analyze only the files that are built by these custom commands.

If `autobuild` fails, or you want to analyze a different set of source files from those built by the `autobuild` process, you'll need to remove the `autobuild` step from the workflow, and manually add build steps. For C/C++, C#, Go, Kotlin, Java, and Swift projects, CodeQL will analyze whatever source code is built by your specified build steps. For more information about how to configure CodeQL code scanning for compiled languages, see "CodeQL code scanning for compiled languages."

Uploading code scanning data to GitHub

GitHub can display code analysis data generated externally by a third-party tool. You can upload code analysis data with the `upload-sarif` action. For more information, see "Uploading a SARIF file to GitHub."