About the dependency graph

In this article

You can use the dependency graph to identify all your project's dependencies. The dependency graph supports a range of popular package ecosystems.

About the dependency graph

The dependency graph is a summary of the manifest and lock files stored in a repository and any dependencies that are submitted for the repository using the Dependency submission API (beta). For each repository, it shows:

- Dependencies, the ecosystems and packages it depends on
- Dependents, the repositories and packages that depend on it

For each dependency, you can see the license information and vulnerability severity. You can also search for a specific dependency using the search bar. Dependencies are sorted automatically by vulnerability severity.

When you push a commit to GitHub Enterprise Cloud that changes or adds a supported manifest or lock file to the default branch, the dependency graph is automatically updated. In addition, the graph is updated when anyone pushes a change to the repository of one of your dependencies. For information on the supported ecosystems and manifest files, see "Supported package ecosystems" below.

Additionally, you can use the Dependency submission API (beta) to submit dependencies from the package manager or ecosystem of your choice, even if the ecosystem is not supported by dependency graph for manifest or lock file analysis. Dependencies submitted to a project using the Dependency submission API (beta) will show which detector was used for their submission and when they were submitted. For more information on the Dependency submission API, see "Using the Dependency submission API."

When you create a pull request containing changes to dependencies that targets the default branch, GitHub uses the dependency graph to add dependency reviews to the pull request. These indicate whether the dependencies contain vulnerabilities and, if so, the version of the dependency in which the vulnerability was fixed. For more information, see "About dependency review."

If you have at least read access to the repository, you can export the dependency graph for the repository as an SPDX-compatible, Software Bill of Materials (SBOM), via the

GitHub UI or GitHub REST API. For more information, see "Exporting a software bill of materials for your repository."

Dependency graph availability

The dependency graph is automatically generated for all public repositories. You can choose to enable it for forks and for private repositories. For more information, see "Managing security and analysis settings for your repository."

Repository administrators can also set up the dependency graph for private repositories. For more information, see "Configuring the dependency graph."

Dependencies included

The dependency graph includes all the dependencies of a repository that are detailed in the manifest and lock files, or their equivalent, for supported ecosystems, as well as any dependencies that are submitted using the Dependency submission API (beta). This includes:

- Direct dependencies, that are explicitly defined in a manifest or lock file or have been submitted using the Dependency submission API (beta)
- Indirect dependencies of these direct dependencies, also known as transitive dependencies or sub-dependencies

The dependency graph identifies indirect dependencies only if they are defined in a lock file or have been submitted using the Dependency submission API (beta). For the most reliable graph, you should use lock files (or their equivalent) because they define exactly which versions of the direct and indirect dependencies you currently use. If you use lock files, you also ensure that all contributors to the repository are using the same versions, which will make it easier for you to test and debug code. If your ecosystem does not have lock files, you can use pre-made actions that resolve transitive dependencies for many ecosystems. For more information, see "Using the Dependency submission API."

For more information on how GitHub Enterprise Cloud helps you understand the dependencies in your environment, see "About supply chain security."

Dependents included

For public repositories, only public repositories that depend on it or on packages that it publishes are reported. This information is not reported for private repositories.

Using the dependency graph

You can use the dependency graph to:

- Explore the repositories your code depends on, and those that depend on it. For more information, see "Exploring the dependencies of a repository."
- View a summary of the dependencies used in your organization's repositories in a single dashboard. For more information, see "Viewing insights for your organization."
- View and update vulnerable dependencies for your repository. For more information, see "About Dependabot alerts."
- See information about vulnerable dependencies in pull requests. For more information, see "Reviewing dependency changes in a pull request."

Supported package ecosystems

The recommended formats explicitly define which versions are used for all direct and all indirect dependencies. If you use these formats, your dependency graph is more accurate. It also reflects the current build set up and enables the dependency graph to report vulnerabilities in both direct and indirect dependencies. Indirect dependencies that are inferred from a manifest file (or equivalent) are excluded from the checks for insecure dependencies.

Package manager	Languages	Recommended formats	All supported formats
Cargo	Rust	`Cargo.lock`	`Cargo.toml`, `Cargo.lock`
Composer	PHP	`composer.lock`	`composer.json`, `composer.lock`
NuGet	.NET languages (C#, F#, VB), C++	`.csproj`, `.vbproj`, `.nuspec`, `.vcxproj`, `.fsproj`	`.csproj`, `.vbproj`, `.nuspec`, `.vcxproj`, `.fsproj`, `packages.config`
GitHub Actions workflows	YAML	`.yml`, `.yaml`	`.yml`, `.yaml`
Go modules	Go	`go.mod`	`go.mod`
Maven	Java, Scala	`pom.xml`	`pom.xml`
npm	JavaScript	`package-lock.json`	`package-lock.json`, `package.json`
pip	Python	`requirements.txt`, `pipfile.lock`	`requirements.txt`, `pipfile`, `pipfile.lock`, `setup.py`
pnpm	JavaScript	`pnpm-lock.yaml`	`package.json`, `pnpm-lock.yaml`
pub	Dart	`pubspec.lock`	`pubspec.yaml`, `pubspec.lock`
Python Poetry	Python	`poetry.lock`	`poetry.lock`, `pyproject.toml`
RubyGems	Ruby	`Gemfile.lock`	`Gemfile.lock`, `Gemfile`, `*.gemspec`
Swift Package Manager	Swift	`Package.resolved`	`Package.resolved`
Yarn	JavaScript	`yarn.lock`	`package.json`, `yarn.lock`

Notes:

- If you list your Python dependencies within a `setup.py` file, we may not be able to parse and list every dependency in your project.
- GitHub Actions workflows must be located in the `.github/workflows/` directory of a repository to be recognized as manifests. Any actions or workflows referenced using the syntax `jobs[*].steps[*].uses` or `jobs.<job_id>.uses` will be parsed as dependencies. For more information, see "Workflow syntax for GitHub Actions."

- Dependabot will only create Dependabot alerts for vulnerable GitHub Actions that use semantic versioning. You will not receive alerts for a vulnerable action that uses SHA versioning. If you use GitHub Actions with SHA versioning, we recommend enabling Dependabot version updates for your repository or organization to keep the actions you use updated to the latest versions. For more information, see "About Dependabot alerts" and "About Dependabot version updates."

You can use the Dependency submission API (beta) to add dependencies from the package manager or ecosystem of your choice to the dependency graph, even if the ecosystem is not in the supported ecosystem list above. Dependencies submitted to a project using the Dependency submission API (beta) will show which detector was used for their submission and when they were submitted.

You will only get Dependabot alerts for dependencies that are from one of the supported ecosystems of the GitHub Advisory Database. For more information on the Dependency submission API, see "Using the Dependency submission API."

Further reading

- "Dependency graph" on Wikipedia
- "Exploring the dependencies of a repository"
- "Viewing and updating Dependabot alerts"
- "Troubleshooting the detection of vulnerable dependencies"