Troubleshooting the dependency graph

In this article

If the dependency information reported by the dependency graph is not what you expected, there are a number of points to consider, and various things you can check.

The results of dependency detection reported by GitHub Enterprise Cloud may be different from the results returned by other tools. There are good reasons for this and it's helpful to understand how GitHub determines dependencies for your project.

Does the dependency graph only find dependencies in manifests and lockfiles?

The dependency graph automatically includes information on dependencies that are explicitly declared in your environment. That is, dependencies that are specified in a manifest or a lockfile. The dependency graph generally also includes transitive dependencies, even when they aren't specified in a lockfile, by looking at the dependencies of the dependencies in a manifest file.

The dependency graph doesn't automatically include "loose" dependencies. "Loose" dependencies are individual files that are copied from another source and checked into the repository directly or within an archive (such as a ZIP or JAR file), rather than being referenced by in a package manager’s manifest or lockfile.

However, you can use the Dependency submission API (beta) to add dependencies to a project's dependency graph, even if the dependencies are not declared in a manifest or lock file, such as dependencies resolved when a project is built. 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."

Check: Is the missing dependency for a component that's not specified in the repository's manifest or lockfile?

Does the dependency graph detect dependencies specified using variables?

The dependency graph analyzes manifests as they’re pushed to GitHub. The dependency graph doesn't, therefore, have access to the build environment of the project, so it can't resolve variables used within manifests. If you use variables within a manifest to specify the name, or more commonly the version of a dependency, then that dependency will not automatically be included in the dependency graph.

However, you can use the Dependency submission API (beta) to add dependencies to a project's dependency graph, even if the dependencies are only resolved when a project is built. For more information on the Dependency submission API, see "Using the Dependency submission API."

Check: Is the missing dependency declared in the manifest by using a variable for its name or version?

Are there limits which affect the dependency graph data?

Yes, the dependency graph has one category of limits:

1. **Processing limits**

   These affect the dependency graph displayed within GitHub and also prevent Dependabot alerts being created.

   Manifests over 0.5 MB in size are only processed for enterprise accounts. For other accounts, manifests over 0.5 MB are ignored and will not create Dependabot alerts.

   By default, GitHub will not process more than 150 manifests per repository. Dependabot alerts are not created for manifests beyond this limit. If you need to increase the limit, you can contact us through the GitHub Support portal.

   Manifest files stored in directories with names that are typically used for vendored dependencies will not be processed. A directory whose name matches the following regular expressions is considered a vendored dependencies directory:

   - `(3rd|[Tt]hird)[-_]?[Pp]arty/`
   - `(^|/)vendors?/`
   - `(^|/)[Ee]xtern(als?)?/`
   - `(^|/)[Vv]+endor/`

   Examples:

   - third-party/dependencies/dependency1
   - vendors/dependency1
   - /externals/vendor1/dependency1

Further reading

- "About the dependency graph"
- "Managing security and analysis settings for your repository"
- "Troubleshooting the detection of vulnerable dependencies"
- "Troubleshooting Dependabot errors"