Configuration options for the dependabot.yml file

In this article

Detailed information for all the options you can use to customize how Dependabot maintains your repositories.

Who can use this feature
People with write permissions to a repository can configure Dependabot for the repository.

About the `dependabot.yml` file

The Dependabot configuration file, `dependabot.yml`, uses YAML syntax. If you're new to YAML and want to learn more, see "Learn YAML in five minutes."

You must store this file in the `.github` directory of your repository in the default branch. When you add or update the `dependabot.yml` file, this triggers an immediate check for version updates. For more information and an example, see "Configuring Dependabot version updates."

Any options that also affect security updates are used the next time a security alert triggers a pull request for a security update. For more information, see "Configuring Dependabot security updates."

Note: You cannot configure Dependabot alerts using the `dependabot.yml` file.

The `dependabot.yml` file has two mandatory top-level keys: `version`, and `updates`. You can, optionally, include a top-level `registries` key. The file must start with `version: 2`.

Configuration options for the `dependabot.yml` file

The top-level `updates` key is mandatory. You use it to configure how Dependabot updates the versions or your project's dependencies. Each entry configures the update settings for a particular package manager. You can use the following options.

Option	Required	Security Updates	Version Updates	Description
`package-ecosystem`	✓	✕	✓	Package manager to use
`directory`	✓	✓	✓	Location of package

				manifests
`schedule.interval`	✓	✕	✓	How often to check for updates
`allow`	✕	✓	✓	Customize which updates are allowed
`assignees`	✕	✓	✓	Assignees to set on pull requests
`commit-message`	✕	✓	✓	Commit message preferences
`enable-beta-ecosystems`	✕	✕	✓	Enable ecosystems that have beta-level support
`groups`	✕	✕	✓	Group updates for certain dependencies
`ignore`	✕	See `ignore`	See `ignore`	Ignore certain dependencies or versions
`insecure-external-code-execution`	✕	✓	✓	Allow or deny code execution in manifest files
`labels`	✕	✓	✓	Labels to set on pull requests
`milestone`	✕	✓	✓	Milestone to set on pull requests
`open-pull-requests-limit`	✕	✕	✓	Limit number of open pull requests for version updates
`pull-request-branch-name.separator`	✕	✓	✓	Change separator for pull request branch names
`rebase-strategy`	✕	✓	✓	Disable automatic rebasing
`registries`	✕	✓	✓	Private registries that Dependabot can access
`reviewers`	✕	✓	✓	Reviewers to set on pull requests
`schedule.day`	✕	✕	✓	Day of week to check for updates
`schedule.time`	✕	✕	✓	Time of day to check for updates (hh:mm)

`schedule.timezone`	✗	✗	✓	Timezone for time of day (zone identifier)
`target-branch`	✗	✗	✓	Branch to create pull requests against
`vendor`	✗	✓	✓	Update vendored or cached dependencies
`versioning-strategy`	✗	✓	✓	How to update manifest version requirements

These options fit broadly into the following categories.

- Essential set up options that you must include in all configurations: `package-ecosystem` , `directory` , `schedule.interval` .
- Options to customize the update schedule: `schedule.time` , `schedule.timezone` , `schedule.day` .
- Options to control which dependencies are updated: `allow` , `groups` , `ignore` , `vendor` .
- Options to add metadata to pull requests: `reviewers` , `assignees` , `labels` , `milestone` .
- Options to change the behavior of the pull requests: `target-branch` , `versioning-strategy` , `commit-message` , `rebase-strategy` , `pull-request-branch-name.separator` .

In addition, the `open-pull-requests-limit` option changes the maximum number of pull requests for version updates that Dependabot can open.

> **Note:** Some of these configuration options may also affect pull requests raised for security updates of vulnerable package manifests.
>
> Security updates are raised for vulnerable package manifests only on the default branch. When configuration options are set for the same branch (true unless you use `target-branch`), and specify a `package-ecosystem` and `directory` for the vulnerable manifest, then pull requests for security updates use relevant options.
>
> In general, security updates use any configuration options that affect pull requests, for example, adding metadata or changing their behavior. For more information about security updates, see "Configuring Dependabot security updates."

`package-ecosystem`

Required. You add one `package-ecosystem` element for each package manager that you want Dependabot to monitor for new versions. The repository must also contain a dependency manifest or lock file for each of these package managers. If you want to enable vendoring for a package manager that supports it, the vendored dependencies must be located in the required directory. For more information, see `vendor` below.

The following table shows, for each package manager:

- The YAML value to use in the `dependabot.yml` file
- The supported versions of the package manager
- Whether dependencies in private GitHub repositories or registries are supported
- Whether vendored dependencies are supported

Package manager	YAML value	Supported versions	Private repositories	Private registries	Vendoring

Bundler	`bundler`	v1, v2	✗	✓	✓
Cargo	`cargo`	v1	✓	✓ (git only)	✗
Composer	`composer`	v1, v2	✓	✓	✗
Docker	`docker`	v1	✓	✓	Not applicable
Hex	`mix`	v1	✗	✓	✗
elm-package	`elm`	v0.19	✓	✓	✗
git submodule	`gitsubmodule`	Not applicable	✓	✓	Not applicable
GitHub Actions	`github-actions`	Not applicable	✓	✓	Not applicable
Go modules	`gomod`	v1	✓	✓	✓
Gradle	`gradle`	Not applicable	✓	✓	✗
Maven	`maven`	Not applicable	✓	✓	✗
npm	`npm`	v6, v7, v8, v9	✓	✓	✗
NuGet	`nuget`	<= 4.8	✓	✓	✗
pip	`pip`	v21.1.2	✗	✓	✗
pipenv	`pip`	<= 2021-05-29	✗	✓	✗
pip-compile	`pip`	6.1.0	✗	✓	✗
pnpm	`npm`	v7, v8	✓	✓	✗
poetry	`pip`	v1	✗	✓	✗
pub	`pub`	v2	✗	✗	✗
Swift	`swift`	v5	✓	✓ (git only)	✗
Terraform	`terraform`	>= 0.13, <= 1.5.x	✓	✓	Not applicable
yarn	`npm`	v1, v2, v3	✓	✓	✓

Tip: For package managers such as `pipenv` and `poetry`, you need to use the `pip` YAML value. For example, if you use `poetry` to manage your Python dependencies and want Dependabot to monitor your dependency manifest file for new versions, use `package-ecosystem: "pip"` in your `dependabot.yml` file.

Cargo

Private registry support applies to git registries, and doesn't include cargo registries.

Docker

Dependabot can add metadata from Docker images to pull requests for version updates.

The metadata includes release notes, changelogs and the commit history. Repository administrators can use the metadata to quickly evaluate the stability risk of the dependency update.

In order for Dependabot to fetch Docker metadata, maintainers of Docker images must add the `org.opencontainers.image.source` label to their Dockerfile, and include the URL of the source repository. Additionally, maintainers must tag the repository with the same tags as the published Docker images. For an example, see the `dependabot-fixtures/docker-with-source` repository. For more information on Docker labels, see Extension image labels and BUILDX_GIT_LABELS in the Docker documentation.

Dependabot can update Docker image tags in Kubernetes manifests. Add an entry to the Docker `package-ecosystem` element of your `dependabot.yml` file for each directory containing a Kubernetes manifest which references Docker image tags. Kubernetes manifests can be Kubernetes Deployment YAML files or Helm charts. For information about configuring your `dependabot.yml` file for `docker`, see "`package-ecosystem`" in "Configuration options for the dependabot.yml file."

Dependabot supports both public and private Docker registries. For a list of the supported registries, see "`docker-registry`" in "Configuration options for the dependabot.yml file."

GitHub Actions

Dependabot only supports updates to GitHub Actions using the GitHub repository syntax, such as actions/checkout@v4. Docker Hub and GitHub Packages Container registry URLs are currently not supported.

Dependabot supports both public and private repositories for GitHub Actions. For private registry configuration options, see "`git`" in "Configuration options for the dependabot.yml file."

Gradle

Dependabot doesn't run Gradle but supports updates to the following files:

- `build.gradle`, `build.gradle.kts` (for Kotlin projects)
- `gradle/libs.versions.toml` (for projects using a standard Gradle version catalog)
- Files included via the `apply` declaration that have `dependencies` in the filename. Note that `apply` does not support `apply to`, recursion, or advanced syntaxes (for example, Kotlin's `apply` with `mapOf`, filenames defined by property).

For Dependabot security updates, Gradle support is limited to manual uploads of the dependency graph data using the dependency submission API. For more information about the dependency submission API, see "Using the Dependency submission API."

Note: When you upload Gradle dependencies to the dependency graph using the dependency submission API, all project dependencies are uploaded, even indirect dependencies that aren't explicitly mentioned in any dependency file. When an alert is detected in an indirect dependency, Dependabot isn't able to find the vulnerable dependency in the repository, and therefore won't create a security update for that alert.

Maven

Dependabot doesn't run Maven but supports updates to `pom.xml` files.

NuGet CLI

Dependabot doesn't run the NuGet CLI but does support most features up until version

4.8.

pip and pip-compile

In addition to supporting updates to `requirements.txt` files, Dependabot supports updates to `pyproject.toml` files if they follow the PEP 621 standard.

pnpm

pnpm is supported for Dependabot version updates and Dependabot security updates.

pub

Dependabot won't perform an update for `pub` when the version that it tries to update to is ignored, even if an earlier version is available.

Swift

Private registry support applies to git registries only. Swift registries are not supported. Non-declarative manifests are not supported. For more information on non-declarative manifests, see Editing Non-Declarative Manifests in the Swift Evolution documentation.

yarn

Dependabot supports vendored dependencies for v2 onwards.

Example of a basic setup for three package managers

```yaml
# Basic set up for three package managers

version: 2
updates:

  # Maintain dependencies for GitHub Actions
  - package-ecosystem: "github-actions"
    # Workflow files stored in the default location of `.github/workflows`. (You
don't need to specify `/.github/workflows` for `directory`. You can use
`directory: "/"`.)
    directory: "/"
    schedule:
      interval: "weekly"

  # Maintain dependencies for npm
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"

  # Maintain dependencies for Composer
  - package-ecosystem: "composer"
    directory: "/"
    schedule:
      interval: "weekly"
```

`directory`

Required. You must define the location of the package manifests for each package manager (for example, the *package.json* or *Gemfile*). You define the directory relative to the root of the repository for all ecosystems except GitHub Actions.

For GitHub Actions, you do not need to set the directory to `/.github/workflows`. Configuring the key to `/` automatically instructs Dependabot to search the `/.github/workflows` directory, as well as the *action.yml* / *action.yaml* file from the root directory.

```yaml
# Specify location of manifest files for each package manager

version: 2
updates:
  - package-ecosystem: "composer"
    # Files stored in repository root
    directory: "/"
    schedule:
      interval: "weekly"

  - package-ecosystem: "npm"
    # Files stored in `app` directory
    directory: "/app"
    schedule:
      interval: "weekly"

  - package-ecosystem: "github-actions"
    # Workflow files stored in the default location of `.github/workflows`. (You don't need to specify `/.github/workflows` for `directory`. You can use `directory: "/"`.)
    directory: "/"
    schedule:
      interval: "weekly"
```

`schedule.interval`

Required. You must define how often to check for new versions for each package manager. By default, Dependabot randomly assigns a time to apply all the updates in the configuration file. To set a specific time, you can use `schedule.time` and `schedule.timezone`.

Note: The `schedule.time` option is a best effort, and it may take some time before Dependabot opens pull requests to update to newer dependency versions.

Interval types	Frequency
`daily`	Runs on every weekday, Monday to Friday.
`weekly`	Runs once each week. By default, this is on Monday. To modify this, use `schedule.day`.
`monthly`	Runs once each month. This is on the first day of the month.

```yaml
# Set update schedule for each package manager

version: 2
updates:

  - package-ecosystem: "github-actions"
    # Workflow files stored in the default location of `.github/workflows`. (You don't need to specify `/.github/workflows` for `directory`. You can use `directory: "/"`.)
    directory: "/"
    schedule:
      # Check for updates to GitHub Actions every weekday
      interval: "daily"
  - package-ecosystem: "composer"
    directory: "/"
    schedule:
      # Check for updates managed by Composer once a week
      interval: "weekly"
```

Note: `schedule` defines when Dependabot attempts a new update. However, it's not the only time you may receive pull requests. Updates can be triggered based on changes to your `dependabot.yml` file, changes to your manifest file(s) after a failed update, or Dependabot security updates. For more information, see "About Dependabot version updates" and "About Dependabot security updates."

Sometimes, due to a misconfiguration or an incompatible version, you might see that a Dependabot run has failed. After 30 failed runs, Dependabot version updates will skip subsequent scheduled runs until you manually trigger a check for updates from the dependency graph, or you update the manifest file. Dependabot security updates will still run as usual.

`allow`

By default all dependencies that are explicitly defined in a manifest are kept up to date by Dependabot version updates. In addition, Dependabot security updates also update vulnerable dependencies that are defined in lock files. You can use `allow` and `ignore` to customize which dependencies to maintain. Dependabot checks for all allowed dependencies and then filters out any ignored dependencies or versions. So a dependency that is matched by both an `allow` and an `ignore` will be ignored.

Use the `allow` option to customize which dependencies are updated. This applies to both version and security updates. You can use the following options:

- `dependency-name`—use to allow updates for dependencies with matching names, optionally using `*` to match zero or more characters.
  - For Java dependencies, the format of the `dependency-name` attribute is: `groupId:artifactId`; for example: `org.kohsuke:github-api`.
  - For Docker image tags, the format is the full name of the repository; for example, for an image tag of `<account ID>.dkr.ecr.us-west-2.amazonaws.com/base/foo/bar/ruby:3.1.0-focal-jemalloc`, use `base/foo/bar/ruby`.
- `dependency-type`—use to allow updates for dependencies of specific types.

Dependency types	Supported by package managers	Allow updates
`direct`	All	All explicitly defined dependencies.
`indirect`	`bundler`, `pip`, `composer`, `cargo`, `gomod`	Dependencies of direct dependencies (also known as sub-dependencies, or transient dependencies).
`all`	All	All explicitly defined dependencies. For `bundler`, `pip`, `composer`, `cargo`, `gomod`, also the dependencies of direct dependencies.
`production`	`bundler`, `composer`, `mix`, `maven`, `npm`, `pip`	Only dependencies in the "Production dependency group".

`development`	`bundler`, `composer`, `mix`, `maven`, `npm`, `pip`	Only dependencies in the "Development dependency group".

```yaml
# Use `allow` to specify which dependencies to maintain

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    allow:
      # Allow updates for Lodash
      - dependency-name: "lodash"
      # Allow updates for React and any packages starting "react"
      - dependency-name: "react*"

  - package-ecosystem: "composer"
    directory: "/"
    schedule:
      interval: "weekly"
    allow:
      # Allow both direct and indirect updates for all packages
      - dependency-type: "all"

  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
    allow:
      # Allow only direct updates for
      # Django and any packages starting "django"
      - dependency-name: "django*"
        dependency-type: "direct"
      # Allow only production updates for Sphinx
      - dependency-name: "sphinx"
        dependency-type: "production"
```

`assignees`

Use `assignees` to specify individual assignees for all pull requests raised for a package manager.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```yaml
# Specify assignees for pull requests

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    # Add assignees
    assignees:
      - "octocat"
```

`commit-message`

By default, Dependabot attempts to detect your commit message preferences and use similar patterns. Use the `commit-message` option to specify your preferences explicitly.

Supported options

Note: The `prefix` and the `prefix-development` options have a 50 character limit.

- `prefix` specifies a prefix for all commit messages. When you specify a prefix for commit messages, GitHub will automatically add a colon between the defined prefix and the commit message provided the defined prefix ends with a letter, number, closing parenthesis, or closing bracket. This means that, for example, if you end the prefix with a whitespace, there will be no colon added between the prefix and the commit message. The code snippet below provides examples of both in the same configuration file.
- `prefix-development` specifies a separate prefix for all commit messages that update dependencies in the Development dependency group. When you specify a value for this option, the `prefix` is used only for updates to dependencies in the Production dependency group. This is supported by: `bundler`, `composer`, `mix`, `maven`, `npm`, and `pip`.
- `include: "scope"` specifies that any prefix is followed by a list of the dependencies updated in the commit.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```yaml
# Customize commit messages

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    commit-message:
      # Prefix all commit messages with "npm: "
      prefix: "npm"

  - package-ecosystem: "docker"
    directory: "/"
    schedule:
      interval: "weekly"
    commit-message:
      # Prefix all commit messages with "[docker] " (no colon, but a trailing
whitespace)
      prefix: "[docker] "

  - package-ecosystem: "composer"
    directory: "/"
    schedule:
      interval: "weekly"
    # Prefix all commit messages with "Composer" plus its scope, that is, a
    # list of updated dependencies
    commit-message:
      prefix: "Composer"
      include: "scope"

  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
    # Include a list of updated dependencies
    # with a prefix determined by the dependency group
    commit-message:
      prefix: "pip prod"
      prefix-development: "pip dev"
      include: "scope"
```

If you use the same configuration as in the example above, bumping the `requests` library in the `pip` development dependency group will generate a commit message of:

`pip dev: bump requests from 1.0.0 to 1.0.1`

`groups`

You can only create groups for Dependabot version updates. Dependabot security updates do not support grouped updates. In addition, if there is a grouped pull request for a vulnerable package, Dependabot security updates will always attempt to create a separate pull request, even if the existing group pull request is an update to the same, or a later, version.

By default, Dependabot raises a single pull request for each dependency that needs to be updated to a newer version. You can use `groups` to create sets of dependencies (per package manager), so that Dependabot opens a single pull request to update multiple dependencies at the same time.

You can also specify grouping settings based on how updates affect a specific ecosystem and follow semantic versioning (SemVer). This means you can, for example, group all patch updates together. This approach helps Dependabot create as few pull requests as possible, while also reducing the chances of accidentally accepting changes that could cause issues. If a package follows SemVer, there's a higher chance (but not a guarantee) that minor and patch updates will be backwards compatible.

> **Note:** SemVer is an accepted standard for defining versions of software packages, in the form `x.y.z`. Dependabot assumes that versions in this form are always `major.minor.patch`.

When you first configure a group, you specify a group name that will display in pull request titles and branch names. You can then define other options to include or exclude specific dependencies from the group. You must use the `patterns`, `dependency-type`, or `update-types` options to define the group, or any combination thereof.

Option	Description
`dependency-type`	Use to specify a dependency type to be included in the group. `dependency-type` can be `development` or `production`.
`patterns`	Use to define strings of characters that match with a dependency name (or multiple dependency names) to include those dependencies in the group.
`exclude-patterns`	Use to exclude certain dependencies from the group. If a dependency is excluded from a group, Dependabot will continue to raise single pull requests to update the dependency to its latest version.
`update-types`	Use to specify the semantic versioning level to include in the group. Possible values are `minor`, `patch`, and `major`.

Example 1

The `dependabot.yml` file configuration uses `patterns` and `dependency-type` options to include specific dependencies in the group, and `exclude-patterns` to exclude a dependency (or multiple dependencies) from the group.

```yaml
# `dependabot.yml` file using the `dependency-type` option to group updates
# in conjunction with `patterns` and `exclude-patterns`.

groups:
  production-dependencies:
    dependency-type: "production"
  development-dependencies:
    dependency-type: "development"
    exclude-patterns:
    - "rubocop*"
  rubocop:
    patterns:
    - "rubocop*"
```

Example 2

A `dependabot.yml` file with a customized Bundler configuration, which has been modified to create a group of dependencies. The configuration specifies `patterns` (strings of characters) that match with the name of a dependency (or multiple dependencies) in order to include the dependencies in the group.

```yaml
# `dependabot.yml` file with customized Bundler configuration
# In this example, the name of the group is `dev-dependencies`, and
# only the `patterns` and `exclude-patterns` options are used.
version: 2
updates:
  # Keep bundler dependencies up to date
  - package-ecosystem: "bundler"
    directory: "/"
    schedule:
      interval: "weekly"
    # Create a group of dependencies to be updated together in one pull request
    groups:
       # Specify a name for the group, which will be used in pull request titles
       # and branch names
       dev-dependencies:
          # Define patterns to include dependencies in the group (based on
          # dependency name)
          patterns:
            - "rubocop" # A single dependency name
            - "rspec*"  # A wildcard string that matches multiple dependency
names
            - "*"       # A wildcard that matches all dependencies in the package
                        # ecosystem. Note: using "*" may open a large pull
request
          # Define patterns to exclude dependencies from the group (based on
          # dependency name)
          exclude-patterns:
            - "gc_ruboconfig"
            - "gocardless-*"
```

Example 3

The `dependabot.yml` file is configured so that any packages matching the pattern `@angular*` where the highest resolvable version is `minor` or `patch` will be grouped together. Dependabot will create a separate pull request for any package that doesn't match the pattern, or that doesn't update to a `minor` or `patch` version.

```yaml
# `dependabot.yml` file using the `update-types` option to group updates.
# Any packages matching the pattern @angular* where the highest resolvable
# version is minor or patch will be grouped together.
version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    groups:
      angular:
        patterns:
        - "@angular*"
        update-types:
        - "minor"
        - "patch"
```

Example 4

The `dependabot.yml` file uses an `ignore` condition to exclude updates to `major` versions of `@angular*` packages.

```
# `dependabot.yml` file using the `update-types` option to group updates
# in conjunction with an `ignore` condition.
# If you do not want updates to `major` versions of `@angular*` packages, you can
specify an `ignore` condition
groups:
  angular:
    patterns:
    - "@angular*"
    update-types:
    - "minor"
    - "patch"
ignore:
  - dependency-name: "@angular*"
    update-types: ["version-update:semver-major"]
```

Dependabot creates groups in the order they appear in your `dependabot.yml` file. If a dependency update could belong to more than one group, it is only assigned to the first group it matches with.

If a dependency doesn't belong to any group, Dependabot will continue to raise single pull requests to update the dependency to its latest version as normal. GitHub reports in the logs if a group is empty. For more information, see "Dependabot fails to group a set of dependencies into a single pull request."

When a scheduled update runs, Dependabot will refresh pull requests for grouped updates using the following rules:

- If all the same dependencies need to be updated to the same versions, Dependabot will rebase the branch.
- If all the same dependencies need to be updated, but a newer version has become available for one (or more) of the dependencies, Dependabot will close the pull request and create a new one.
- If the dependencies to be updated have changed - for example, if another dependency in the group now has an update available - Dependabot will close the pull request and create a new one.

You can also manage pull requests for grouped version updates using comment commands, which are short comments you can make on a pull request to give instructions to Dependabot. For more information, see "Managing pull requests for dependency updates."

`ignore`

By default all dependencies that are explicitly defined in a manifest are kept up to date by Dependabot version updates. In addition, Dependabot security updates also update vulnerable dependencies that are defined in lock files. You can use `allow` and `ignore` to customize which dependencies to maintain. Dependabot checks for all allowed dependencies and then filters out any ignored dependencies or versions. So a dependency that is matched by both an `allow` and an `ignore` will be ignored.

Dependencies can be ignored either by adding them to `ignore` or by using the `@dependabot ignore` command on a pull request opened by Dependabot.

Creating `ignore` conditions from `@dependabot ignore`

Dependencies ignored by using the `@dependabot ignore` command are stored centrally for each package manager. If you start ignoring dependencies in the `dependabot.yml` file, these existing preferences are considered alongside the `ignore` dependencies in the configuration.

You can check whether a repository has stored `ignore` preferences by searching the repository for `"@dependabot ignore" in:comments`, or by using the `@dependabot show DEPENDENCY_NAME ignore conditions` comment command. If you wish to unblock updates for a dependency ignored this way, re-open the pull request. This clears the `ignore` conditions that were set when the pull request was closed and resumes those Dependabot version updates for the dependency. To update the dependency to a newer version, merge the pull request. In pull requests for grouped version updates, you can also use the `@dependabot unignore` commands to clear `ignore` settings for dependencies.

For more information about the `@dependabot ignore` commands, see "Managing pull requests for dependency updates."

Specifying dependencies and versions to ignore

You can use the `ignore` option to customize which dependencies are updated. The `ignore` option supports the following options.

Option	Description
`dependency-name`	Use to ignore updates for dependencies with matching names, optionally using `*` to match zero or more characters. For Java dependencies, the format of the `dependency-name` attribute is: `groupId:artifactId` (for example: `org.kohsuke:github-api`). To prevent Dependabot from automatically updating TypeScript type definitions from DefinitelyTyped, use `@types/*`.
`versions`	Use to ignore specific versions or ranges of versions. If you want to define a range, use the standard pattern for the package manager. For example, for npm, use `^1.0.0`; for Bundler, use `~> 2.0`; for Docker, use Ruby version syntax; for NuGet, use `7.*`.
`update-types`	Use to ignore types of updates, such as semver `major`, `minor`, or `patch` updates on version updates (for example: `version-update:semver-patch` will ignore patch updates). You can combine this with `dependency-name: "*"` to ignore particular `update-types` for all dependencies. Currently, `version-update:semver-major`,

`version-update:semver-minor`, and `version-update:semver-patch` are the only supported options.

When used alone, the `ignore.versions` key affects both Dependabot updates, but the `ignore.update-types` key affects only Dependabot version updates.

However, if `versions` and `update-types` are used together in the same `ignore` rule, both Dependabot updates are affected, unless the configuration uses `target-branch` to check for version updates on a non-default branch.

```
# Use `ignore` to specify dependencies that should not be updated

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    ignore:
      - dependency-name: "express"
        # For Express, ignore all Dependabot updates for version 4 and 5
        versions: ["4.x", "5.x"]
        # For Lodash, ignore all updates
      - dependency-name: "lodash"
        # For AWS SDK, ignore all patch updates for version updates only
      - dependency-name: "aws-sdk"
        update-types: ["version-update:semver-patch"]
```

Note: Dependabot can only run version updates on manifest or lock files if it can access all of the dependencies in the file, even if you add inaccessible dependencies to the `ignore` option of your configuration file. For more information, see "Managing security and analysis settings for your organization" and "Troubleshooting Dependabot errors."

Note: For the `pub` ecosystem, Dependabot won't perform an update when the version that it tries to update to is ignored, even if an earlier version is available.

`insecure-external-code-execution`

Package managers with the `package-ecosystem` values `bundler`, `mix`, and `pip` may execute external code in the manifest as part of the version update process. This might allow a compromised package to steal credentials or gain access to configured registries. When you add a `registries` setting within an `updates` configuration, Dependabot automatically prevents external code execution, in which case the version update may fail. You can choose to override this behavior and allow external code execution for `bundler`, `mix`, and `pip` package managers by setting `insecure-external-code-execution` to `allow`.

```
# Allow external code execution when updating dependencies from private
registries

version: 2
registries:
  ruby-github:
    type: rubygems-server
    url: https://rubygems.pkg.github.com/octocat/github_api
    token: ${{secrets.MY_GITHUB_PERSONAL_TOKEN}}
updates:
  - package-ecosystem: "bundler"
    directory: "/rubygems-server"
    insecure-external-code-execution: allow
    registries: "*"
    schedule:
      interval: "monthly"
```

If you define a `registries` setting to allow Dependabot to access a private package registry, and you set `insecure-external-code-execution` to `allow` in the same `updates` configuration, external code execution that occurs will only have access to the package managers in the registries associated with that `updates` setting. There is no access allowed to any of the registries defined in the top level `registries` configuration.

In this example, the configuration file allows Dependabot to access the `ruby-github` private package registry. In the same `updates` setting, `insecure-external-code-execution` is set to `allow`, which means that the code executed by dependencies will only access the `ruby-github` registry, and not the `dockerhub` registry.

```
# Using `registries` in conjunction with `insecure-external-code-execution:allow`
# in the same `updates` setting

version: 2
registries:
  ruby-github:
    type: rubygems-server
    url: https://rubygems.pkg.github.com/octocat/github_api
    token: ${{secrets.MY_GITHUB_PERSONAL_TOKEN}}
  dockerhub:
    type: docker-registry
    url: registry.hub.docker.com
    username: octocat
    password: ${{secrets.DOCKERHUB_PASSWORD}}
updates:
  - package-ecosystem: "bundler"
    directory: "/rubygems-server"
    insecure-external-code-execution: allow
    registries:
      - ruby-github # only access to registries associated with this
ecosystem/directory
    schedule:
      interval: "monthly"
```

You can explicitly deny external code execution, regardless of whether there is a `registries` setting for this update configuration, by setting `insecure-external-code-execution` to `deny`.

`labels`

By default, Dependabot raises all pull requests with the `dependencies` label. If more than one package manager is defined, Dependabot includes an additional label on each pull request. This indicates which language or ecosystem the pull request will update, for example: `java` for Gradle updates and `submodules` for git submodule updates. Dependabot creates these default labels automatically, as necessary in your repository.

Use `labels` to override the default labels and specify alternative labels for all pull requests raised for a package manager. If any of these labels is not defined in the repository, it is ignored. To disable all labels, including the default labels, use `labels: [ ]`.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```
# Specify labels for pull requests

version: 2
```

```
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    # Specify labels for npm pull requests
    labels:
      - "npm"
      - "dependencies"
```

`milestone`

Use `milestone` to associate all pull requests raised for a package manager with a milestone. You need to specify the numeric identifier of the milestone and not its label. If you view a milestone, the final part of the page URL, after `milestone`, is the identifier. For example: `https://github.com/<org>/<repo>/milestone/3`.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```
# Specify a milestone for pull requests

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    # Associate pull requests with milestone "4"
    milestone: 4
```

`open-pull-requests-limit`

By default, Dependabot opens a maximum of five pull requests for version updates. Once there are five open pull requests from Dependabot, Dependabot will not open any new requests until some of those open requests are merged or closed. Use `open-pull-requests-limit` to change this limit. This also provides a simple way to temporarily disable version updates for a package manager.

This option has no impact on security updates, which have a separate, internal limit of ten open pull requests.

```
# Specify the number of open pull requests allowed

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    # Disable version updates for npm dependencies
    open-pull-requests-limit: 0

  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
    # Allow up to 10 open pull requests for pip dependencies
    open-pull-requests-limit: 10
```

`pull-request-branch-name.separator`

Dependabot generates a branch for each pull request. Each branch name includes `dependabot`, and the package manager and dependency that are updated. By default, these parts are separated by a `/` symbol, for example: `dependabot/npm_and_yarn/next_js/acorn-6.4.1`.

Use `pull-request-branch-name.separator` to specify a different separator. This can be one of: `"-"`, `_` or `/`. The hyphen symbol must be quoted because otherwise it's interpreted as starting an empty YAML list.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```yaml
# Specify a different separator for branch names

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    pull-request-branch-name:
      # Separate sections of the branch name with a hyphen
      # for example, `dependabot-npm_and_yarn-next_js-acorn-6.4.1`
      separator: "-"
```

`rebase-strategy`

By default, Dependabot automatically rebases open pull requests when it detects any changes to the pull request. Use `rebase-strategy` to disable this behavior.

Note: If a pull request has not been merged for 30 days, Dependabot will stop rebasing the pull request. You can still manually rebase and merge the pull request.

Available rebase strategies

- `auto` to use the default behavior and rebase open pull requests when changes are detected.
- `disabled` to disable automatic rebasing.

When `rebase-strategy` is set to `auto`, Dependabot attempts to rebase pull requests in the following cases.

- When you use Dependabot version updates, for any open Dependabot pull request when your schedule runs.
- When you reopen a closed Dependabot pull request.
- When you change the value of `target-branch` in the Dependabot configuration file. For more information about this field, see "`target-branch`."
- When Dependabot detects that a Dependabot pull request is in conflict after a recent push to the target branch.

When `rebase-strategy` is set to `disabled`, Dependabot stops rebasing pull requests.

Note: This behavior only applies to pull requests that go into conflict with the target branch. Dependabot will keep rebasing (until 30 days after opening) pull requests opened prior to the `rebase-strategy` setting being changed, and pull requests that are part of a scheduled run.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```yaml
# Disable automatic rebasing

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
    # Disable rebasing for npm pull requests
    rebase-strategy: "disabled"
```

`registries`

To allow Dependabot to access a private package registry when performing a version update, you must include a `registries` setting within the relevant `updates` configuration. You can allow all of the defined registries to be used by setting `registries` to `"*"`. Alternatively, you can list the registries that the update can use. To do this, use the name of the registry as defined in the top-level `registries` section of the `dependabot.yml` file. For more information, see "Configuration options for private registries" below.

For in-depth information about available options, as well as recommendations and advice when configuring private registries, see "Guidance for the configuration of private registries for Dependabot."

To allow Dependabot to use `bundler`, `mix`, and `pip` package managers to update dependencies in private registries, you can choose to allow external code execution. For more information, see `insecure-external-code-execution` above.

```yaml
# Allow Dependabot to use one of the two defined private registries
# when updating dependency versions for this ecosystem

version: 2
registries:
  maven-github:
    type: maven-repository
    url: https://maven.pkg.github.com/octocat
    username: octocat
    password: ${{secrets.MY_ARTIFACTORY_PASSWORD}}
  npm-npmjs:
    type: npm-registry
    url: https://registry.npmjs.org
    username: octocat
    password: ${{secrets.MY_NPM_PASSWORD}}
updates:
  - package-ecosystem: "gitsubmodule"
    directory: "/"
    registries:
      - maven-github
    schedule:
      interval: "monthly"
```

`reviewers`

Use `reviewers` to specify individual reviewers or teams of reviewers for all pull requests raised for a package manager. You must use the full team name, including the organization, as if you were @mentioning the team.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

```
# Specify reviewers for pull requests

version: 2
updates:
  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
    # Add reviewers
    reviewers:
      - "octocat"
      - "my-username"
      - "my-org/python-team"
```

`schedule.day`

When you set a `weekly` update schedule, by default, Dependabot checks for new versions on Monday at a random set time for the repository. Use `schedule.day` to specify an alternative day to check for updates.

Supported values

- `monday`
- `tuesday`
- `wednesday`
- `thursday`
- `friday`
- `saturday`
- `sunday`

```
# Specify the day for weekly checks

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
      # Check for npm updates on Sundays
      day: "sunday"
```

`schedule.time`

By default, Dependabot checks for new versions at a random set time for the repository. Use `schedule.time` to specify an alternative time of day to check for updates (format: `hh:mm`).

```
# Set a time for checks
version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
      # Check for npm updates at 9am UTC
      time: "09:00"
```

`schedule.timezone`

By default, Dependabot checks for new versions at a random set time for the repository. Use `schedule.timezone` to specify an alternative time zone. The time zone identifier must be from the Time Zone database maintained by iana. For more information, see List of tz database time zones.

```
# Specify the timezone for checks

version: 2
updates:
  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
      time: "09:00"
      # Use Japan Standard Time (UTC +09:00)
      timezone: "Asia/Tokyo"
```

`target-branch`

By default, Dependabot checks for manifest files on the default branch and raises pull requests for version updates against this branch. Use `target-branch` to specify a different branch for manifest files and for pull requests. When you use this option, the settings for this package manager will no longer affect any pull requests raised for security updates.

```
# Specify a non-default branch for pull requests for pip

version: 2
updates:
  - package-ecosystem: "pip"
    directory: "/"
    schedule:
      interval: "weekly"
    # Raise pull requests for version updates
    # to pip against the `develop` branch
    target-branch: "develop"
    # Labels on pull requests for version updates only
    labels:
      - "pip dependencies"

  - package-ecosystem: "npm"
    directory: "/"
    schedule:
      interval: "weekly"
      # Check for npm updates on Sundays
      day: "sunday"
    # Labels on pull requests for security and version updates
    labels:
      - "npm dependencies"
```

`vendor`

Use the `vendor` option to tell Dependabot to vendor dependencies when updating them. Don't use this option if you're using `gomod` as Dependabot automatically detects vendoring for this tool.

```
# Configure version updates for both dependencies defined in manifests and
vendored dependencies

version: 2
```

```
updates:
  - package-ecosystem: "bundler"
    # Raise pull requests to update vendored dependencies that are checked in to
the repository
    vendor: true
    directory: "/"
    schedule:
      interval: "weekly"
```

Dependabot only updates the vendored dependencies located in specific directories in a repository.

Package manager	Required file path for vendored dependencies	More information
`bundler`	The dependencies must be in the *vendor/cache* directory. Other file paths are not supported.	`bundle cache` documentation
`gomod`	No path requirement (dependencies are usually located in the *vendor* directory)	`go mod vendor` documentation

`versioning-strategy`

When Dependabot edits a manifest file to update a version, there are several different potential versioning strategies:

Option	Action
`auto`	Try to differentiate between apps and libraries. Use `increase` for apps and `widen` for libraries.
`increase`	Always increase the minimum version requirement to match the new version. If a range already exists, typically this only increases the lower bound.
`increase-if-necessary`	Leave the constraint if the original constraint allows the new version, otherwise, bump the constraint.
`lockfile-only`	Only create pull requests to update lockfiles. Ignore any new versions that would require package manifest changes.
`widen`	Widen the allowed version requirements to include both the new and old versions, when possible. Typically, this only increases the maximum allowed version requirement.
N/A	Some package managers do not yet support configuring the `versioning-strategy` parameter.

The following table shows an example of how `versioning-strategy` can be used.

Current constraint	Current version	New version	Strategy	New constraint
^1.0.0	1.0.0	1.2.0	`widen`	^1.0.0

^1.0.0	1.0.0	1.2.0	`increase`	^1.2.0
^1.0.0	1.0.0	1.2.0	`increase-if-necessary`	^1.0.0
^1.0.0	1.0.0	2.0.0	`widen`	>=1.0.0 <3.0.0
^1.0.0	1.0.0	2.0.0	`increase`	^2.0.0
^1.0.0	1.0.0	2.0.0	`increase-if-necessary`	^2.0.0

Use the `versioning-strategy` option to change this behavior for supported package managers.

Setting this option will also affect pull requests for security updates to the manifest files of this package manager, unless you use `target-branch` to check for version updates on a non-default branch.

Available update strategies:

Ecosystem	**Supported versioning strategies**	**Default strategy**
`bundler`	`auto`, `increase`, `increase-if-necessary`, `lockfile-only`	`auto`
`cargo`	`auto`, `lockfile-only`	`auto`
`composer`	`auto`, `increase`, `increase-if-necessary`, `lockfile-only`, `widen`	`auto`
`docker`	N/A	N/A
`github-actions`	N/A	N/A
`gitsubmodule`	N/A	N/A
`gomod`	N/A	N/A
`gradle`	N/A	N/A
`maven`	N/A	N/A
`mix`	`auto`, `lockfile-only`	`auto`
`npm`	`auto`, `increase`, `increase-if-necessary`, `lockfile-only`, `widen`	`auto`
`nuget`	N/A	N/A
`pip`	`auto`, `increase`, `increase-if-necessary`, `lockfile-only`	`auto`
`pub`	`auto`, `increase`, `increase-if-necessary`, `widen`	`auto`
`terraform`	N/A	N/A

Note: `N/A` indicates that the package manager does not yet support configuring the `versioning-strategy` parameter. The strategy code is open source, so if you'd like a particular ecosystem to support a new strategy, you are always welcome to submit a pull request in https://github.com/dependabot/dependabot-core/.

```yaml
# Example configuration for customizing the manifest version strategy

version: 2
updates:
  - package-ecosystem: "composer"
    directory: "/"
    schedule:
      interval: "weekly"
    # Increase the version requirements for Composer only when required
    versioning-strategy: increase-if-necessary
```

Configuration options for private registries

The top-level `registries` key is optional. It allows you to specify authentication details that Dependabot can use to access private package registries.

Note: Private registries behind firewalls on private networks are not supported.

The value of the `registries` key is an associative array, each element of which consists of a key that identifies a particular registry and a value which is an associative array that specifies the settings required to access that registry. The following `dependabot.yml` file configures a registry identified as `dockerhub` in the `registries` section of the file and then references this in the `updates` section of the file.

```yaml
# Minimal settings to update dependencies in one private registry

version: 2
registries:
  dockerhub: # Define access for a private registry
    type: docker-registry
    url: registry.hub.docker.com
    username: octocat
    password: ${{secrets.DOCKERHUB_PASSWORD}}
updates:
  - package-ecosystem: "docker"
    directory: "/docker-registry/dockerhub"
    registries:
      - dockerhub # Allow version updates for dependencies in this registry
    schedule:
      interval: "monthly"
```

You use the following options to specify access settings. Registry settings must contain a `type` and a `url`, and typically either a `username` and `password` combination or a `token`.

Option	Description
`type`	Identifies the type of registry. See the full list of types below.
`url`	The URL to use to access the dependencies in this registry. The protocol is optional. If not specified, `https://` is assumed. Dependabot adds or ignores trailing slashes as required.

`username`	The username that Dependabot uses to access the registry. `username` is the username or email address for the account.
`password`	A reference to a Dependabot secret containing the password for the specified user. For more information, see "Configuring access to private registries for Dependabot." `password` is the password for the account specified by the username. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.
`key`	A reference to a Dependabot secret containing an access key for this registry. For more information, see "Configuring access to private registries for Dependabot."
`token`	A reference to a Dependabot secret containing an access token for this registry. For more information, see "Configuring access to private registries for Dependabot." `token` is used to provide an access token for an external system and should not be used to provide a GitHub personal access token. If you want to use a GitHub personal access token, you should supply it as a password.
`replaces-base`	For registries, if the boolean value is `true`, Dependabot will resolve dependencies by using the specified URL rather than the base URL of that specific ecosystem. For example, for registries with `type: python-index`, if the boolean value is `true`, pip resolves dependencies by using the specified URL rather than the base URL of the Python Package Index (by default `https://pypi.org/simple`).

You must provide the required settings for each configuration `type` that you specify. Some types allow more than one way to connect. The following sections provide details of the settings you should use for each `type`.

For in-depth information about available options, as well as recommendations and advice when configuring private registries, see "Guidance for the configuration of private registries for Dependabot."

`composer-repository`

The `composer-repository` type supports username and password. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

This registry type will prefix-match the path provided in the `url` option. This means you can provide multiple credentials to the same host, which can be used to access distinct paths. However, if you don't have multiple registries on the same host, we recommend that you omit the path from the `url`, so that all paths to the registry will receive credentials.

```yaml
registries:
  composer:
    type: composer-repository
    url: https://repo.packagist.com/example-company/
    username: octocat
    password: ${{secrets.MY_PACKAGIST_PASSWORD}}
```

`docker-registry`

Dependabot works with any container registries that implement the OCI container registry spec. For more information, see https://github.com/opencontainers/distribution-spec/blob/main/spec.md. Dependabot supports authentication to private registries via a central token service or HTTP Basic Auth. For further details, see Token Authentication Specification in the Docker documentation and Basic access authentication on Wikipedia.

The `docker-registry` type supports username and password. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

This registry type will prefix-match the path provided in the `url` option. This means you can provide multiple credentials to the same host, which can be used to access distinct paths. However, if you don't have multiple registries on the same host, we recommend that you omit the path from the `url`, so that all paths to the registry will receive credentials.

```
registries:
  dockerhub:
    type: docker-registry
    url: https://registry.hub.docker.com
    username: octocat
    password: ${{secrets.MY_DOCKERHUB_PASSWORD}}
    replaces-base: true
```

The `docker-registry` type can also be used to pull from private Amazon ECR using static AWS credentials.

```
registries:
  ecr-docker:
    type: docker-registry
    url: https://1234567890.dkr.ecr.us-east-1.amazonaws.com
    username: ${{secrets.ECR_AWS_ACCESS_KEY_ID}}
    password: ${{secrets.ECR_AWS_SECRET_ACCESS_KEY}}
    replaces-base: true
```

`git`

The `git` type supports username and password. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

```
registries:
  github-octocat:
    type: git
    url: https://github.com
    username: x-access-token
    password: ${{secrets.MY_GITHUB_PERSONAL_TOKEN}}
```

`hex-organization`

The `hex-organization` type supports organization and key.

This registry type will prefix-match the path provided in the `url` option. This means you can provide multiple credentials to the same host, which can be used to access distinct paths. However, if you don't have multiple registries on the same host, we recommend that you omit the path from the `url`, so that all paths to the registry will receive credentials.

```
registries:
  github-hex-org:
    type: hex-organization
    organization: github
    key: ${{secrets.MY_HEX_ORGANIZATION_KEY}}
```

`hex-repository`

The `hex-repository` type supports an authentication key.

`repo` is a required field, which must match the name of the repository used in your dependency declaration.

The `public-key-fingerprint` is an optional configuration field, representing the fingerprint of the public key for the Hex repository. `public-key-fingerprint` is used by Hex to establish trust with the private repository. The `public-key-fingerprint` field can be either listed in plaintext or stored as a Dependabot secret.

```
registries:
   github-hex-repository:
     type: hex-repository
     repo: private-repo
     url: https://private-repo.example.com
     auth-key: ${{secrets.MY_AUTH_KEY}}
     public-key-fingerprint: ${{secrets.MY_PUBLIC_KEY_FINGERPRINT}}
```

`maven-repository`

The `maven-repository` type supports username and password. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

This registry type will prefix-match the path provided in the `url` option. This means you can provide multiple credentials to the same host, which can be used to access distinct paths. However, if you don't have multiple registries on the same host, we recommend that you omit the path from the `url`, so that all paths to the registry will receive credentials.

```
registries:
  maven-artifactory:
    type: maven-repository
    url: https://acme.jfrog.io/artifactory/my-maven-registry
    username: octocat
    password: ${{secrets.MY_ARTIFACTORY_PASSWORD}}
```

`npm-registry`

The `npm-registry` type supports username and password, or token. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

When using username and password, your `.npmrc`'s auth token may contain a `base64` encoded `_password`; however, the password referenced in your Dependabot configuration file must be the original (unencoded) password.

Note: When using `npm.pkg.github.com`, don't include a path. Instead use the `https://npm.pkg.github.com` URL without a path.

```
registries:
  npm-npmjs:
    type: npm-registry
    url: https://registry.npmjs.org
    username: octocat
    password: ${{secrets.MY_NPM_PASSWORD}}  # Must be an unencoded password
    replaces-base: true
```

```
registries:
  npm-github:
    type: npm-registry
    url: https://npm.pkg.github.com
    token: ${{secrets.MY_GITHUB_PERSONAL_TOKEN}}
    replaces-base: true
```

For security reasons, Dependabot does not set environment variables. Yarn (v2 and later) requires that any accessed environment variables are set. When accessing environment variables in your `.yarnrc.yml` file, you should provide a fallback value such as `${ENV_VAR-fallback}` or `${ENV_VAR:-fallback}`. For more information, see [Yarnrc files](#) in the Yarn documentation.

`nuget-feed`

The `nuget-feed` type supports username and password, or token. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

```
registries:
  nuget-example:
    type: nuget-feed
    url: https://nuget.example.com/v3/index.json
    username: octocat@example.com
    password: ${{secrets.MY_NUGET_PASSWORD}}
```

```
registries:
  nuget-azure-devops:
    type: nuget-feed
    url: https://pkgs.dev.azure.com/.../_packaging/My_Feed/nuget/v3/index.json
    username: octocat@example.com
    password: ${{secrets.MY_AZURE_DEVOPS_TOKEN}}
```

`python-index`

The `python-index` type supports username and password, or token. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

This registry type will prefix-match the path provided in the `url` option. This means you can provide multiple credentials to the same host, which can be used to access distinct paths. However, if you don't have multiple registries on the same host, we recommend that you omit the path from the `url`, so that all paths to the registry will receive credentials.

```
registries:
  python-example:
    type: python-index
    url: https://example.com/_packaging/my-feed/pypi/example
    username: octocat
    password: ${{secrets.MY_BASIC_AUTH_PASSWORD}}
    replaces-base: true
```

```
registries:
  python-azure:
    type: python-index
    url: https://pkgs.dev.azure.com/octocat/_packaging/my-feed/pypi/example
    username: octocat@example.com
    password: ${{secrets.MY_AZURE_DEVOPS_TOKEN}}
    replaces-base: true
```

`rubygems-server`

The `rubygems-server` type supports username and password, or token. If the account is a GitHub account, you can use a GitHub personal access token in place of the password.

This registry type will prefix-match the path provided in the `url` option. This means you can provide multiple credentials to the same host, which can be used to access distinct paths. However, if you don't have multiple registries on the same host, we recommend that you omit the path from the `url`, so that all paths to the registry will receive credentials.

```
registries:
  ruby-example:
    type: rubygems-server
    url: https://rubygems.example.com
    username: octocat@example.com
    password: ${{secrets.MY_RUBYGEMS_PASSWORD}}
    replaces-base: true
```

```
registries:
  ruby-github:
    type: rubygems-server
    url: https://rubygems.pkg.github.com/octocat/github_api
    token: ${{secrets.MY_GITHUB_PERSONAL_TOKEN}}
    replaces-base: true
```

`terraform-registry`

The `terraform-registry` type supports a token.

```
registries:
  terraform-example:
    type: terraform-registry
    url: https://terraform.example.com
    token: ${{secrets.MY_TERRAFORM_API_TOKEN}}
```

Enabling support for beta-level ecosystems

`enable-beta-ecosystems`

By default, Dependabot updates the dependency manifests and lock files only for fully supported ecosystems. Use the `enable-beta-ecosystems` flag to opt in to updates for ecosystems that are not yet generally available.

There are currently no ecosystems in beta.

```
# Configure beta ecosystem

version: 2
enable-beta-ecosystems: true
updates:
  - package-ecosystem: "beta-ecosystem"
```

```
    directory: "/"
    schedule:
      interval: "weekly"
```

Legal