Using the REST API to interact with checks

In this article

You can use the REST API to build GitHub Apps that run powerful checks against code changes in a repository. You can create apps that perform continuous integration, code linting, or code scanning services and provide detailed feedback on commits.

Overview

Rather than binary pass/fail build statuses, GitHub Apps can report rich statuses, annotate lines of code with detailed information, and re-run tests. REST API to manage checks is available exclusively to your GitHub Apps.

For an example of how to use the REST API with a GitHub App, see "Building CI checks with a GitHub App."

You can use statuses with protected branches to prevent people from merging pull requests prematurely. For more information, see "About protected branches."

About check suites

When someone pushes code to a repository, GitHub creates a check suite for the last commit. A check suite is a collection of the check runs created by a single GitHub App for a specific commit. Check suites summarize the status and conclusion of the check runs that a suite includes.

The `status` can be `queued`, `in_progress`, or `completed`.

If the status is `completed`, the conclusion can be any of the following:

- `action_required`
- `cancelled`
- `timed_out`
- `failure`
- `neutral`
- `skipped`
- `stale`
- `startup_failure`

- `success`

The check suite reports the highest priority check run `conclusion` in the check suite's `conclusion`. For example, if three check runs have conclusions of `timed_out`, `success`, and `neutral` the check suite conclusion will be `timed_out`.

By default, GitHub creates a check suite automatically when code is pushed to the repository. This default flow sends the `check_suite` event (with `requested` action) to all GitHub Apps that have the `checks:write` permission. When your GitHub App receives the `check_suite` event, it can create new check runs for the latest commit. GitHub automatically adds new check runs to the correct check suite based on the check run's repository and SHA.

If you don't want to use the default automatic flow, you can control when you create check suites. To change the default settings for the creation of check suites, use the Update repository preferences for check suites endpoint. All changes to the automatic flow settings are recorded in the audit log for the repository. If you have disabled the automatic flow, you can create a check suite using the Create a check suite endpoint. You should continue to use the Create a check run endpoint to provide feedback on a commit.

Write permission for the REST API to interact with checks is only available to GitHub Apps. OAuth apps and authenticated users can view check runs and check suites, but they are not able to create them. If you aren't building a GitHub App, you might be interested in the using the REST API to interact with commit statuses.

To use the endpoints to manage check suites, the GitHub App must have the `checks:write` permission and can also subscribe to the check_suite webhook.

For information on how to authenticate as a GitHub App, see "About authentication with a GitHub App."

About check runs

A check run is an individual test that is part of a check suite. Each run includes a status and conclusion.

The status can be `queued`, `in_progress`, or `completed`.

If the status is `completed`, the conclusion can be any of the following:

- `action_required`
- `cancelled`
- `timed_out`
- `failure`
- `neutral`
- `skipped`
- `success`

If a check run is in an incomplete state for more than 14 days, then the check run's `conclusion` becomes `stale` and appears on GitHub as stale with . Only GitHub can mark check runs as `stale`. For more information about possible conclusions of a check run, see the `conclusion` parameter.

As soon as you receive the `check_suite` webhook, you can create the check run, even if the check is not complete. You can update the `status` of the check run as it completes with the values `queued`, `in_progress`, or `completed`, and you can update the `output` as more details become available. A check run can contain timestamps, a link to more details on your external site, detailed annotations for specific lines of code, and information about the analysis performed.

Annotations add information from your check run to specific lines of code. Each annotation includes an `annotation_level` property, which can be `notice`, `warning`, or `failure`. The annotation also includes `path`, `start_line`, and `end_line` to specify what location the annotation refers to. The annotation includes a `message` to describe the result. For more information, see "Check Runs" in the REST API reference documentation.

A check can also be manually re-run in the GitHub UI. See "About status checks" for more details. When this occurs, the GitHub App that created the check run will receive the `check_run` webhook requesting a new check run. If you create a check run without creating a check suite, GitHub creates the check suite for you automatically.

Write permission for the REST API to interact with checks is only available to GitHub Apps. OAuth apps and authenticated users can view check runs and check suites, but they are not able to create them. If you aren't building a GitHub App, you might be interested in the using the REST API to interact with commit statuses.

To use the endpoints to manage check runs, the GitHub App must have the `checks:write` permission and can also subscribe to the check_run webhook.

Check runs and requested actions

When you set up a check run with requested actions (not to be confused with GitHub Actions), you can display a button in the pull request view on GitHub that allows people to request your GitHub App to perform additional tasks.

For example, a code linting app could use requested actions to display a button in a pull request to automatically fix detected syntax errors.

To create a button that can request additional actions from your app, use the `actions` object when you Create a check run. For example, the `actions` object below displays a button in the **Checks** tab of a pull request with the label "Fix this." The button appears after the check run completes.

```
"actions": [{
  "label": "Fix this",
  "description": "Let us fix that for you",
  "identifier": "fix_errors"
}]
```

When a user clicks the button, GitHub sends the `check_run.requested_action` webhook to your app. When your app receives a `check_run.requested_action` webhook event, it can look for the `requested_action.identifier` key in the webhook payload to determine which button was clicked and perform the requested task.

For a detailed example of how to set up requested actions with the REST API, see "Building CI checks with a GitHub App."

Retention of checks data

GitHub.com retains checks data for 400 days. After 400 days, the data is archived. 10 days after archival, the data is permanently deleted.

To merge a pull request with checks that are both required and archived, you must rerun the checks.