Using bundlesize with Travis CI

- Set the performance budget
  - Create a test script
- Set up continuous integration
  - Integrate GitHub and Travis CI
  - Authorize bundlesize to post on pull requests
- Try it out
  - Trigger your first bundlesize test
  - Optimize
  - Re-run test
  - Monitor

Using bundlesize with Travis CI lets you define performance budgets with minimal setup and enforce them as part of your development workflow. Travis CI is a service that runs tests for your app in the cloud every time you push code to GitHub. You can configure your repository so that it won't allow merging pull-requests unless the bundlesize tests have passed.

Bundlesize's GitHub checks include a size comparison to the main branch and a warning in case of a big jump in size.

Bundlesize check on GitHub

You can also use bundlesize with Circle CI, Wrecker and Drone.

To see it in action, here's an app bundled with webpack that lets you vote for your favorite kitty.

Cat voting app

Set the performance budget

This Glitch already contains bundlesize.

The main bundle of this app is in the public folder. To test its size, add the following section to the `package.json` file:

```
"bundlesize": [
  {
    "path": "./public/*.bundle.js",
    "maxSize": "170 kB"
  }
]
```

You can also set different thresholds for different files. This is especially useful if you are splitting a bundle in your application.

To keep the compressed JavaScript bundle size under the recommended limit, set the performance budget to 170KB in the `maxSize` field.

Bundlesize supports glob patterns and the * wildcard character in the file path will match all bundle names in the public folder.

By default, bundlesize tests gzipped sizes. You can use the compression option to switch to brotli compression or turn it off completely.

Create a test script

Since Travis needs a test to run, add a test script to `package.json`:

```
"scripts": {
  "start": "webpack && http-server -c-1",
  "test": "bundlesize"
}
```

Set up continuous integration

Integrate GitHub and Travis CI

First, create a new repository for this project on your GitHub account and initialize it with a `README.md`.

You'll need to register an account on Travis and activate GitHub Apps integration under the Settings section of your profile.

GitHub Apps integration on Travis CI

Once you have an account, go to **Settings** under your profile, click the **Sync account** button, and make sure your new repo is listed on Travis.

Travis CI Sync button

Authorize bundlesize to post on pull requests

Bundlesize needs authorization to be able to post on pull requests, so visit this link to get the bundlesize token that will be stored in the Travis configuration.

bundlesize token

In your project's Travis dashboard, go to **More options** > **Settings** > **Environment variables**.

Adding environment variables on Travis CI

Add a new environment variable with the token as the value field and BUNDLESIZE_GITHUB_TOKEN as the name.

The last thing you need to kick-off continuous integration is a `.travis.yml` file, which tells Travis CI what to do. To speed things up, it is already included in the project and it specifies that the app is using NodeJS.

With this step, you're all set up and bundlesize will warn you if your JavaScript ever goes over the budget. Even when you start off great, over time, as you add new features, kilobytes can pile up. With automated performance budget monitoring, you can rest easy knowing that it won't go unnoticed.

Try it out

Trigger your first bundlesize test

To see how the app stacks up against the performance budget, add the code to the GitHub repo that you created in step 3.

1. On Glitch, click **Tools** > **Git, Import, and Export** > **Export to GitHub**.
2. In the pop-up, enter your GitHub username and the name of the repo as `username/repo`. Glitch will export your app to a new branch named "glitch".
3. Create a new pull request by clicking the **New pull request** button on the homepage of the repository.

You'll now see status checks in progress on the pull request page.

GitHub checks in progress

It won't take long until all checks are done. Unfortunately, the cat voting app is a bit bloated and does not pass the performance budget check. The main bundle is 266 KB and the budget is 170 KB.

Failed bundlesize check

Optimize

Luckily, there are some easy performance wins you can make by removing unused code. There are two main imports in `src/index.js`:

```
import firebase from "firebase";
import * as moment from 'moment';
```

The app is using Firebase Realtime Database to store the data, but it's importing the entire firebase package which consists of a lot more than just a database (auth, storage, messaging etc.).

Fix this by importing only the package that the app needs in the `src/index.js` file:

```
import firebase from "firebase";
import firebase from 'firebase/app';
import 'firebase/database';
```

The `firebase/app` import, which sets up the API surface for each of the different services, is always required.

Re-run test

Since the source file has been updated, you need to run webpack to build the new bundle file.

1. Click the **Tools** button.
2. Then click the **Console** button. This will open the console in another tab.
3. In the console, type `webpack` and wait for it to finish the build.
4. Export the code to GitHub from **Tools** > **Git, Import, and Export** > **Export to GitHub**.
5. Go to the pull request page on GitHub and wait for all checks to finish.

Passed bundlesize check

Success! The new size of the bundle is 125.5 KB and all the checks have passed. □□

Unlike Firebase, importing parts of the moment library cannot be done as easily, but it's worth a shot. Check out how you can further optimize the app in the Remove unused code codelab.

Monitor

The app is now under the budget and all is well. Travis CI and bundlesize will keep monitoring the performance budget for you, making sure your app stays fast.

□□□ #3
□ □□□ □□□ 2026-10-31 22:38:22 UTC
□ cocobook-sync □□□ 2026-11-04 17:38:55 UTC