Building and testing Node.js

In this article

You can create a continuous integration (CI) workflow to build and test your Node.js project.

Note: GitHub-hosted runners are not currently supported on GitHub Enterprise Server. You can see more information about planned future support on the GitHub public roadmap.

Introduction

This guide shows you how to create a continuous integration (CI) workflow that builds and tests Node.js code. If your CI tests pass, you may want to deploy your code or publish a package.

Prerequisites

We recommend that you have a basic understanding of Node.js, YAML, workflow configuration options, and how to create a workflow file. For more information, see:

- "Learn GitHub Actions"
- "Getting started with Node.js"

Using self-hosted runners on GitHub Enterprise Server

When using setup actions (such as `actions/setup-LANGUAGE`) on GitHub Enterprise Server with self-hosted runners, you might need to set up the tools cache on runners that do not have internet access. For more information, see "Setting up the tool cache on self-hosted runners without internet access."

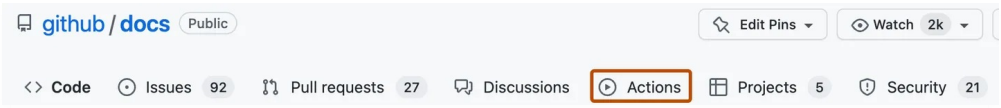
Using a Node.js starter workflow

To get started quickly, add a starter workflow to the `.github/workflows` directory of your repository.

GitHub provides a starter workflow for Node.js that should work for most Node.js projects. The subsequent sections of this guide give examples of how you can customize

this starter workflow.

1. On your GitHub Enterprise Server instance, navigate to the main page of the repository.
2. Under your repository name, click **Actions**.



3. If you already have a workflow in your repository, click **New workflow**.
4. The "Choose a workflow" page shows a selection of recommended starter workflows. Search for "Node.js".
5. Filter the selection of workflows by clicking **Continuous integration**.
6. On the "Node.js" workflow, click **Configure**.

   If you don't find the "Node.js" starter workflow, copy the following workflow code to a new file called `node.js.yml` in the `.github/workflows` directory of your repository.

YAML

```yaml
name: Node.js CI

on:
  push:
    branches: [ "main" ]
  pull_request:
    branches: [ "main" ]

jobs:
  build:
    runs-on: ubuntu-latest

    strategy:
      matrix:
        node-version: [14.x, 16.x, 18.x]
        # See supported Node.js release schedule at
https://nodejs.org/en/about/releases/

    steps:
    - uses: actions/checkout@v4
    - name: Use Node.js ${{ matrix.node-version }}
      uses: actions/setup-node@v3
      with:
        node-version: ${{ matrix.node-version }}
        cache: 'npm'
    - run: npm ci
    - run: npm run build --if-present
    - run: npm test
```

7. Edit the workflow as required. For example, change the Node versions you want to use.
8. Click **Commit changes**.

Specifying the Node.js version

The easiest way to specify a Node.js version is by using the `setup-node` action provided by GitHub. For more information see, `setup-node`.

The `setup-node` action takes a Node.js version as an input and configures that version on the runner. The `setup-node` action finds a specific version of Node.js from the tools cache on each runner and adds the necessary binaries to `PATH`, which persists for the rest of the job. Using the `setup-node` action is the recommended way of using Node.js with GitHub Actions because it ensures consistent behavior across different runners and different versions of Node.js. If you are using a self-hosted runner, you must install Node.js and add it to `PATH`.

The starter workflow includes a matrix strategy that builds and tests your code with the Node.js versions listed in `node-version`. The 'x' in the version number is a wildcard character that matches the latest minor and patch release available for a version. Each version of Node.js specified in the `node-version` array creates a job that runs the same steps.

Each job can access the value defined in the matrix `node-version` array using the `matrix` context. The `setup-node` action uses the context as the `node-version` input. The `setup-node` action configures each job with a different Node.js version before building and testing code. For more information about matrix strategies and contexts, see "Workflow syntax for GitHub Actions" and "Contexts."

YAML

```yaml
strategy:
  matrix:
    node-version: ['14.x', '16.x', '18.x']

steps:
- uses: actions/checkout@v4
- name: Use Node.js ${{ matrix.node-version }}
  uses: actions/setup-node@v3
  with:
    node-version: ${{ matrix.node-version }}
```

Alternatively, you can build and test with exact Node.js versions.

YAML

```yaml
strategy:
  matrix:
    node-version: ['10.17.0', '17.9.0']
```

Or, you can build and test using a single version of Node.js too.

YAML

```yaml
name: Node.js CI

on: [push]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
      - uses: actions/checkout@v4
      - name: Use Node.js
        uses: actions/setup-node@v3
        with:
```

```
          node-version: '20.x'
      - run: npm ci
      - run: npm run build --if-present
      - run: npm test
```

If you don't specify a Node.js version, GitHub uses the environment's default Node.js version. For more information, see "Using GitHub-hosted runners".

Installing dependencies

GitHub-hosted runners have npm and Yarn dependency managers installed. You can use npm and Yarn to install dependencies in your workflow before building and testing your code. The Windows and Linux GitHub-hosted runners also have Grunt, Gulp, and Bower installed.

You can also cache dependencies to speed up your workflow. For more information, see "Caching dependencies to speed up workflows."

Example using npm

This example installs the versions in the *package-lock.json* or *npm-shrinkwrap.json* file and prevents updates to the lock file. Using `npm ci` is generally faster than running `npm install`. For more information, see `npm ci` and "Introducing `npm ci` for faster, more reliable builds."

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Use Node.js
  uses: actions/setup-node@v3
  with:
    node-version: '20.x'
- name: Install dependencies
  run: npm ci
```

Using `npm install` installs the dependencies defined in the *package.json* file. For more information, see `npm install`.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Use Node.js
  uses: actions/setup-node@v3
  with:
    node-version: '20.x'
- name: Install dependencies
  run: npm install
```

Example using Yarn

This example installs the dependencies defined in the *yarn.lock* file and prevents updates to the *yarn.lock* file. For more information, see `yarn install`.

YAML

```yaml
steps:
```

```yaml
- uses: actions/checkout@v4
- name: Use Node.js
  uses: actions/setup-node@v3
  with:
    node-version: '20.x'
- name: Install dependencies
  run: yarn --frozen-lockfile
```

Alternatively, you can install the dependencies defined in the *package.json* file.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Use Node.js
  uses: actions/setup-node@v3
  with:
    node-version: '20.x'
- name: Install dependencies
  run: yarn
```

Example using a private registry and creating the .npmrc file

You can use the `setup-node` action to create a local *.npmrc* file on the runner that configures the default registry and scope. The `setup-node` action also accepts an authentication token as input, used to access private registries or publish node packages. For more information, see `setup-node`.

To authenticate to your private registry, you'll need to store your npm authentication token as a secret. For example, create a repository secret called `NPM_TOKEN`. For more information, see "Using secrets in GitHub Actions."

In the example below, the secret `NPM_TOKEN` stores the npm authentication token. The `setup-node` action configures the *.npmrc* file to read the npm authentication token from the `NODE_AUTH_TOKEN` environment variable. When using the `setup-node` action to create an *.npmrc* file, you must set the `NODE_AUTH_TOKEN` environment variable with the secret that contains your npm authentication token.

Before installing dependencies, use the `setup-node` action to create the *.npmrc* file. The action has two input parameters. The `node-version` parameter sets the Node.js version, and the `registry-url` parameter sets the default registry. If your package registry uses scopes, you must use the `scope` parameter. For more information, see `npm-scope`.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Use Node.js
  uses: actions/setup-node@v3
  with:
    always-auth: true
    node-version: '20.x'
    registry-url: https://registry.npmjs.org
    scope: '@octocat'
- name: Install dependencies
  run: npm ci
  env:
    NODE_AUTH_TOKEN: ${{ secrets.NPM_TOKEN }}
```

The example above creates an *.npmrc* file with the following contents:

```
//registry.npmjs.org/:_authToken=${NODE_AUTH_TOKEN}
@octocat:registry=https://registry.npmjs.org/
always-auth=true
```

Example caching dependencies

You can cache and restore the dependencies using the `setup-node` action.

The following example caches dependencies for npm.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- uses: actions/setup-node@v3
  with:
    node-version: '20'
    cache: 'npm'
- run: npm install
- run: npm test
```

The following example caches dependencies for Yarn.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- uses: actions/setup-node@v3
  with:
    node-version: '20'
    cache: 'yarn'
- run: yarn
- run: yarn test
```

The following example caches dependencies for pnpm (v6.10+).

YAML

```yaml
# This workflow uses actions that are not certified by GitHub.
# They are provided by a third-party and are governed by
# separate terms of service, privacy policy, and support
# documentation.

# NOTE: pnpm caching support requires pnpm version >= 6.10.0

steps:
- uses: actions/checkout@v4
- uses: pnpm/action-setup@0609f0983b7a228f052f81ef4c3d6510cae254ad
  with:
    version: 6.10.0
- uses: actions/setup-node@v3
  with:
    node-version: '20'
    cache: 'pnpm'
- run: pnpm install
- run: pnpm test
```

If you have a custom requirement or need finer controls for caching, you can use the `cache` action. For more information, see "Caching dependencies to speed up workflows."

Building and testing your code

You can use the same commands that you use locally to build and test your code. For example, if you run `npm run build` to run build steps defined in your *package.json* file and `npm test` to run your test suite, you would add those commands in your workflow file.

YAML

```yaml
steps:
- uses: actions/checkout@v4
- name: Use Node.js
  uses: actions/setup-node@v3
  with:
    node-version: '20.x'
- run: npm install
- run: npm run build --if-present
- run: npm test
```

Packaging workflow data as artifacts

You can save artifacts from your build and test steps to view after a job completes. For example, you may need to save log files, core dumps, test results, or screenshots. For more information, see "Storing workflow data as artifacts."

Publishing to package registries

You can configure your workflow to publish your Node.js package to a package registry after your CI tests pass. For more information about publishing to npm and GitHub Packages, see "Publishing Node.js packages."