

Troubleshooting

- [Chrome headless doesn't launch on Windows](#)
- [Chrome headless doesn't launch on UNIX](#)
- [Chrome headless disables GPU compositing](#)
- [Chrome is downloaded but fails to launch on Node.js 14](#)
- [Setting Up Chrome Linux Sandbox](#)
 - [\[recommended\] Enable user namespace cloning](#)
 - [\[alternative\] Setup setuid sandbox](#)
- [Running Puppeteer on Travis CI](#)
- [Running Puppeteer on CircleCI](#)
- [Running Puppeteer in Docker](#)
 - [Running on Alpine](#)
 - [Tips](#)
- [Running Puppeteer in the cloud](#)
 - [Running Puppeteer on Google App Engine](#)
 - [Running Puppeteer on Google Cloud Functions](#)
 - [Running Puppeteer on Google Cloud Run](#)
 - [Running Puppeteer on Heroku](#)
 - [Running Puppeteer on AWS Lambda](#)
 - [Running Puppeteer on AWS EC2 instance running Amazon-Linux](#)
- [Code Transpilation Issues](#)

Chrome headless doesn't launch on Windows

Some [chrome policies](#) might enforce running Chrome/Chromium with certain extensions.

Puppeteer passes `--disable-extensions` flag by default and will fail to launch when such policies are active.

To work around this, try running without the flag:

```
const browser = await puppeteer.launch({
  ignoreDefaultArgs: ['--disable-extensions'],
});
```

Context: [issue 3681](#).

Chrome headless doesn't launch on UNIX

Make sure all the necessary dependencies are installed. You can run `ldd chrome | grep not` on a Linux machine to check which dependencies are missing. The common ones are provided below.

- ▶ Debian (e.g. Ubuntu) Dependencies
- ▶ CentOS Dependencies
- ▶ Check out discussions

Chrome headless disables GPU compositing

Chrome/Chromium requires `--use-gl=egl` to [enable GPU acceleration in headless mode](#).

```
const browser = await puppeteer.launch({
  headless: true,
  args: ['--use-gl=egl'],
});
```

Chrome is downloaded but fails to launch on Node.js 14

If you get an error that looks like this when trying to launch Chromium:

```
(node:15505) UnhandledPromiseRejectionWarning: Error: Failed to launch the browser
process!
spawn /Users/.../node_modules/puppeteer/.local-chromium/mac-756035/chrome-
mac/Chromium.app/Contents/MacOS/Chromium ENOENT
```

This means that the browser was downloaded but failed to be extracted correctly. The most common cause is a bug in Node.js v14.0.0 which broke `extract-zip`, the module Puppeteer uses to extract browser downloads into the right place. The bug was fixed in Node.js v14.1.0, so please make sure you're running that version or higher. Alternatively, if you cannot upgrade, you could downgrade to Node.js v12, but we recommend upgrading when possible.

Setting Up Chrome Linux Sandbox

In order to protect the host environment from untrusted web content, Chrome uses [multiple layers of sandboxing](#). For this to work properly, the host should be configured first. If there's no good sandbox for Chrome to use, it will crash with the error `No usable sandbox!`.

If you **absolutely trust** the content you open in Chrome, you can launch Chrome with the `--no-sandbox` argument:

```
const browser = await puppeteer.launch({
  args: ['--no-sandbox', '--disable-setuid-sandbox'],
});
```

NOTE: Running without a sandbox is **strongly discouraged**. Consider configuring a sandbox instead.

There are 2 ways to configure a sandbox in Chromium.

[recommended] Enable [user namespace cloning](#)

User namespace cloning is only supported by modern kernels. Unprivileged user namespaces are generally fine to enable, but in some cases they open up more kernel attack surface for (unsandboxed) non-root processes to elevate to kernel privileges.

```
sudo sysctl -w kernel.unprivileged_userns_clone=1
```

[alternative] Setup [setuid sandbox](#)


The setuid sandbox comes as a standalone executable and is located next to the Chromium that Puppeteer downloads. It is fine to re-use the same sandbox executable for different Chromium versions, so the following could be done only once per host environment:

```
# cd to the downloaded instance
cd <project-dir-path>/node_modules/puppeteer/.local-chromium/linux-
<revision>/chrome-linux/
sudo chown root:root chrome_sandbox
sudo chmod 4755 chrome_sandbox
# copy sandbox executable to a shared location
sudo cp -p chrome_sandbox /usr/local/sbin/chrome-devel-sandbox
# export CHROME_DEVEL_SANDBOX env variable
export CHROME_DEVEL_SANDBOX=/usr/local/sbin/chrome-devel-sandbox
```

You might want to export the `CHROME_DEVEL_SANDBOX` env variable by default. In this case, add the following to the `~/.bashrc` or `.zshenv`:

```
export CHROME_DEVEL_SANDBOX=/usr/local/sbin/chrome-devel-sandbox
```

Running Puppeteer on Travis CI

 We ran our tests for Puppeteer on Travis CI until v6.0.0 (when we've migrated to GitHub Actions) - see our historical [.travis.yml](#) (v5.5.0) for reference.

Tips-n-tricks:

- [xvfb](#) service should be launched in order to run Chromium in non-headless mode
- Runs on Xenial Linux on Travis by default
- Runs `npm install` by default
- `node_modules` is cached by default

`.travis.yml` might look like this:

```
language: node_js
node_js: node
services: xvfb

script:
  - npm run test
```

Running Puppeteer on CircleCI

Running Puppeteer smoothly on CircleCI requires the following steps:

1. Start with a [NodeJS image](#) in your config like so:

```
docker:
  - image: circleci/node:12 # Use your desired version
    environment:
      NODE_ENV: development # Only needed if puppeteer is in
      `devDependencies`
```

- Dependencies like `libXtst6` probably need to be installed via `apt-get`, so use the [threetreelight/puppeteer](#) orb ([instructions](#)), or paste parts of its [source](#) into your own config.
- Lastly, if you're using Puppeteer through Jest, then you may encounter an error spawning child processes:

```
[00:00.0] jest args: --e2e --spec --max-workers=36
Error: spawn ENOMEM
    at ChildProcess.spawn (internal/child_process.js:394:11)
```

This is likely caused by Jest autodetecting the number of processes on the entire machine (`36`) rather than the number allowed to your container (`2`). To fix this, set `jest --maxWorkers=2` in your test command.

Running Puppeteer in Docker

🔧 We used [Cirrus Ci](#) to run our tests for Puppeteer in a Docker container until v3.0.x - see our historical [Dockerfile.linux \(v3.0.1\)](#) for reference.

Getting headless Chrome up and running in Docker can be tricky. The bundled Chromium that Puppeteer installs is missing the necessary shared library dependencies.

To fix, you'll need to install the missing dependencies and the latest Chromium package in your Dockerfile:

```
FROM node:12-slim

# Install latest chrome dev package and fonts to support major charsets (Chinese,
# Japanese, Arabic, Hebrew, Thai and a few others)
# Note: this installs the necessary libs to make the bundled version of Chromium
# that Puppeteer
# installs, work.
RUN apt-get update \
    && apt-get install -y wget gnupg \
    && wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key
add - \
    && sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable
main" >> /etc/apt/sources.list.d/google.list' \
    && apt-get update \
    && apt-get install -y google-chrome-stable fonts-ipafont-gothic fonts-wqy-zenhei
fonts-thai-tlwg fonts-kacst fonts-freefont-ttf libxss1 \
    --no-install-recommends \
    && rm -rf /var/lib/apt/lists/*

# If running Docker >= 1.13.0 use docker run's --init arg to reap zombie processes,
otherwise
# uncomment the following lines to have `dumb-init` as PID 1
# ADD https://github.com/Yelp/dumb-init/releases/download/v1.2.2/dumb-
init_1.2.2_x86_64 /usr/local/bin/dumb-init
# RUN chmod +x /usr/local/bin/dumb-init
# ENTRYPOINT ["dumb-init", "--"]

# Uncomment to skip the chromium download when installing puppeteer. If you do,
# you'll need to launch puppeteer with:
#     browser.launch({executablePath: 'google-chrome-stable'})
```

```
# ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD true

# Install puppeteer so it's available in the container.
RUN npm init -y && \
  npm i puppeteer \
  # Add user so we don't need --no-sandbox.
  # same layer as npm install to keep re-chowned files from using up several
  hundred MBs more space
  && groupadd -r pptruser && useradd -r -g pptruser -G audio,video pptruser \
  && mkdir -p /home/pptruser/Downloads \
  && chown -R pptruser:pptruser /home/pptruser \
  && chown -R pptruser:pptruser /node_modules \
  && chown -R pptruser:pptruser /package.json \
  && chown -R pptruser:pptruser /package-lock.json

# Run everything after as non-privileged user.
USER pptruser

CMD ["google-chrome-stable"]
```

Build the container:

```
docker build -t puppeteer-chrome-linux .
```

Run the container by passing `node -e "<yourscript.js content as a string>"` as the command:

```
docker run -i --init --rm --cap-add=SYS_ADMIN \
  --name puppeteer-chrome puppeteer-chrome-linux \
  node -e "`cat yoursript.js`"
```

There's a full example at <https://github.com/ebidel/try-puppeteer> that shows how to run this Dockerfile from a webserver running on App Engine Flex (Node).

Running on Alpine

The [newest Chromium package](#) supported on Alpine is 93, which corresponds to [Puppeteer v10.2.0](#).

Example Dockerfile:

```
FROM alpine

# Installs latest Chromium (92) package.
RUN apk add --no-cache \
  chromium \
  nss \
  freetype \
  harfbuzz \
  ca-certificates \
  ttf-freefont \
  nodejs \
  yarn
```

```
...

# Tell Puppeteer to skip installing Chrome. We'll be using the installed package.
ENV PUPPETEER_SKIP_CHROMIUM_DOWNLOAD=true \
    PUPPETEER_EXECUTABLE_PATH=/usr/bin/chromium-browser

# Puppeteer v10.0.0 works with Chromium 92.
RUN yarn add puppeteer@10.0.0

# Add user so we don't need --no-sandbox.
RUN addgroup -S pptruser && adduser -S -G pptruser pptruser \
    && mkdir -p /home/pptruser/Downloads /app \
    && chown -R pptruser:pptruser /home/pptruser \
    && chown -R pptruser:pptruser /app

# Run everything after as non-privileged user.
USER pptruser

...
```

Tips

By default, Docker runs a container with a `/dev/shm` shared memory space 64MB. This is [typically too small](#) for Chrome and will cause Chrome to crash when rendering large pages. To fix, run the container with `docker run --shm-size=1gb` to increase the size of `/dev/shm`. Since Chrome 65, this is no longer necessary. Instead, launch the browser with the `--disable-dev-shm-usage` flag:

```
const browser = await puppeteer.launch({
  args: ['--disable-dev-shm-usage'],
});
```

This will write shared memory files into `/tmp` instead of `/dev/shm`. See crbug.com/736452 for more details.

Seeing other weird errors when launching Chrome? Try running your container with `docker run --cap-add=SYS_ADMIN` when developing locally. Since the Dockerfile adds a `pptr` user as a non-privileged user, it may not have all the necessary privileges.

[dumb-init](#) is worth checking out if you're experiencing a lot of zombies Chrome processes sticking around. There's special treatment for processes with PID=1, which makes it hard to terminate Chrome properly in some cases (e.g. in Docker).

Running Puppeteer in the cloud

Running Puppeteer on Google App Engine

The Node.js runtime of the [App Engine standard environment](#) comes with all system packages needed to run Headless Chrome.

To use `puppeteer`, simply list the module as a dependency in your `package.json` and deploy to Google App Engine. Read more about using `puppeteer` on App Engine by following [the official tutorial](#).

Running Puppeteer on Google Cloud Functions

The Node.js 10 runtime of [Google Cloud Functions](#) comes with all system packages needed to run Headless Chrome.

To use `puppeteer`, simply list the module as a dependency in your `package.json` and deploy your function to Google Cloud Functions using the `nodejs10` runtime.

Running Puppeteer on Google Cloud Run

The default Node.js runtime of [Google Cloud Run](#) does not come with the system packages needed to run Headless Chrome. You will need to set up your own `Dockerfile` and [include the missing dependencies](#).

Running Puppeteer on Heroku

Running Puppeteer on Heroku requires some additional dependencies that aren't included on the Linux box that Heroku spins up for you. To add the dependencies on deploy, add the Puppeteer Heroku buildpack to the list of buildpacks for your app under Settings > Buildpacks.

The url for the buildpack is <https://github.com/jontewks/puppeteer-heroku-buildpack>

Ensure that you're using `'--no-sandbox'` mode when launching Puppeteer. This can be done by passing it as an argument to your `.launch()` call: `puppeteer.launch({ args: ['--no-sandbox'] });`

When you click add buildpack, simply paste that url into the input, and click save. On the next deploy, your app will also install the dependencies that Puppeteer needs to run.

If you need to render Chinese, Japanese, or Korean characters you may need to use a buildpack with additional font files like <https://github.com/CoffeeAndCode/puppeteer-heroku-buildpack>

There's also another [simple guide](#) from @timleland that includes a sample project: <https://timleland.com/headless-chrome-on-heroku/>.

Running Puppeteer on AWS Lambda

AWS Lambda [limits](#) deployment package sizes to ~50MB. This presents challenges for running headless Chrome (and therefore Puppeteer) on Lambda. The community has put together a few resources that work around the issues:

- <https://github.com/alixaxel/chrome-aws-lambda> (kept updated with the latest stable release of puppeteer)
- <https://github.com/adieuadieu/serverless-chrome/blob/HEAD/docs/chrome.md> (serverless plugin - outdated)

Running Puppeteer on AWS EC2 instance running Amazon-Linux

If you are using an EC2 instance running amazon-linux in your CI/CD pipeline, and if you want to run Puppeteer tests in amazon-linux, follow these steps.

1. To install Chromium, you have to first enable `amazon-linux-extras` which comes as part of [EPEL \(Extra Packages for Enterprise Linux\)](#):

```
sudo amazon-linux-extras install epel -y
```

2. Next, install Chromium:

```
sudo yum install -y chromium
```

Now Puppeteer can launch Chromium to run your tests. If you do not enable EPEL and if you continue installing chromium as part of `npm install`, Puppeteer cannot launch Chromium due to unavailability of `libatk-1.0.so.0` and many more packages.

Code Transpilation Issues

If you are using a JavaScript transpiler like babel or TypeScript, calling `evaluate()` with an async function might not work. This is because while puppeteer uses `Function.prototype.toString()` to serialize functions while transpilers could be changing the output code in such a way it's incompatible with puppeteer.

Some workarounds to this problem would be to instruct the transpiler not to mess up with the code, for example, configure TypeScript to use latest ecma version (`"target": "es2018"`). Another workaround could be using string templates instead of functions:

```
await page.evaluate(`(async() => {
  console.log('1');
})() `);
```