Quick Start

Welcome to the quick-start! Below are the top three integration we support, with quick directions on getting up and running with popular methods. See the sidebar for all the other supported API's and REST methods!

Choose your library

- Using Puppeteer
- Using Selenium or WebDriver

Using Puppeteer

Whether your looking to get started, or already have an established codebase, browserless aims to make the transition as easy as possible. At a high-level, you'll need to do the following:

1. Install puppeteer
2. Setup your app
3. Update your app to use browserless

1. Install puppeteer

If you haven't chosen a library yet we highly recommend puppeteer as it's fairly active and has many maintainers. It's also built by the developers of Chrome, so it's one of the highest quality libraries around.

Installing puppeteer

```
$ npm i --save puppeteer
$ yarn add puppeteer
```

In development you'll likely want to install puppeteer's bundled version of Chromium, however you won't need this in production as Chrome is all taken care of for you by browserless. To disable the downloading of puppeteer, simply add an environment variable when installing:

Install for production

You can read more about puppeteer's environment variables here.

2. Setup your app

As an example let's write a screenshot service that takes a picture of the website and downloads it. We'll setup a route on the `/image` path and take a picture of a static webpage (though you can make this configurable if you wish):

Screenshot service

```
const express = require('express');
const puppeteer = require('puppeteer');

const app = express();

app.get('/image', async (req, res) => {
  // puppeteer.launch() => Chrome running locally (on the same hardware)
  let browser = null;

    try {
      browser = await puppeteer.launch();
      const page = await browser.newPage();

      await page.goto('http://www.example.com/');
      const screenshot = await page.screenshot();

      res.end(screenshot, 'binary');
    } catch (error) {
      if (!res.headersSent) {
        res.status(400).send(error.message);
      }
    } finally {
      if (browser) {
        browser.close();
      }
    }
});

app.listen(8080, () => console.log('Listening on PORT: 8080'));
```

With that in place, let's re-work some of our functionality to use browserless in production.

3. Update your app to use browserless.

> the browser running.

Screenshot service with browserless

```
const express = require('express');
const puppeteer = require('puppeteer');
const IS_PRODUCTION = process.env.NODE_ENV === 'production';

const app = express();

const getBrowser = () => IS_PRODUCTION ?

  // Connect to browserless so we don't run Chrome on the same hardware in production
  puppeteer.connect({ browserWSEndpoint: 'wss://chrome.browserless.io?token=YOUR-API-TOKEN' }) :

  // Run the browser locally while in development
  puppeteer.launch();

app.get('/image', async (req, res) => {
  let browser = null;

  try {
    browser = await getBrowser();
    const page = await browser.newPage();

    await page.goto('http://www.example.com/');
    const screenshot = await page.screenshot();

    res.end(screenshot, 'binary');
  } catch (error) {
    if (!res.headersSent) {
      res.status(400).send(error.message);
    }
  } finally {
    if (browser) {
      browser.close();
    }
  }
});

app.listen(8080, () => console.log('Listening on PORT: 8080'));
```

That's it! Now you don't have to worry about bundling Chrome or it's dependencies in production and can continue to develop your application.

Using Selenium or WebDriver

Browserless exposes the WebDriver protocol at https://chrome.browserless.io/webdriver. Updating your tests, continuous-integration tests, or application to use it is as simple as specifying a remote connection.

1. Setup WebDriver.
2. Update the builder to use browserless.

Setup WebDriver.

For the purposes of this example we'll be using the Ruby bindings, but almost every language has an accompanying library for WebDriver. Be sure to consult your runtime's library for more direct instructions.

```
# You must specify --headless and --no-sandbox
options = Selenium::WebDriver::Chrome::Options.new
options.add_argument('--headless')
options.add_argument('--no-sandbox')

# Use the webdriver for going to sites, taking pictures, anything else. Make sure you close the b
driver = Selenium::WebDriver.for :chrome, options: options
```

Once that's setup, be sure to verify your script works locally before proceeding to rule out any unwanted behavior.

Update the builder to use browserless

Once you have script working locally, the next step is to specify browserless as the remote server. This will give you the ability to run multiple browsers in parallel, greatly reducing the amount of time your tests take.

```
# You must specify --headless and --no-sandbox
caps = Selenium::WebDriver::Remote::Capabilities.chrome("goog:chromeOptions" => {
  "args" => [
    "--disable-background-timer-throttling",
    "--disable-backgrounding-occluded-windows",
    "--disable-breakpad",
    "--disable-component-extensions-with-background-pages",
    "--disable-dev-shm-usage",
    "--disable-extensions",
```

```
    "--hide-scrollbars",
    "--metrics-recording-only",
    "--mute-audio",
    "--headless",
    "--no-sandbox"
  ]
})
driver = Selenium::WebDriver.for :remote, url: "https://YOUR_API_TOKEN@chrome.browserless.io/webd

# Use the webdriver for going to sites, taking pictures. Make sure you close the browser when don
driver = Selenium::WebDriver.for :chrome, options: options
```

That's it! Once you've verified it's all working properly, the next step is to conditionally use browserless only when running in a continuous integration environment.

What's next?

There's a lot more that you can configure and tune in browserless to handle the needs of your application. Be sure to read about all the options it exposes and how to get the most out of browserless.

HOW IT WORKS →