

# Development Webserver

---



**Cory House**

@housecor

bitnative.com



# The plan



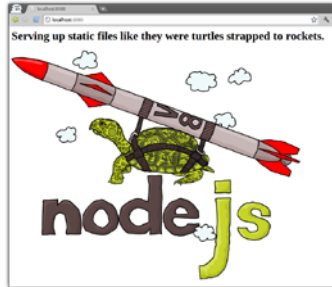
**Review Development webserver options**

**Configure a Dev Webserver**

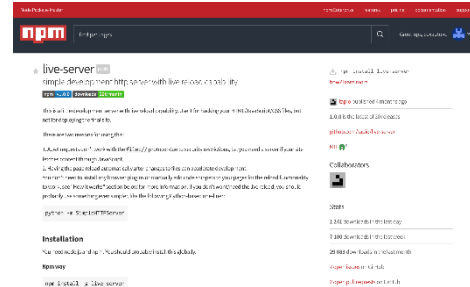
**Services for sharing your work**



# Development Webservers



http-server



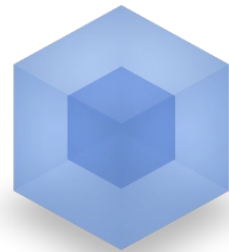
live-server

express

Express

budō

budo



webpack  
MODULE BUNDLER

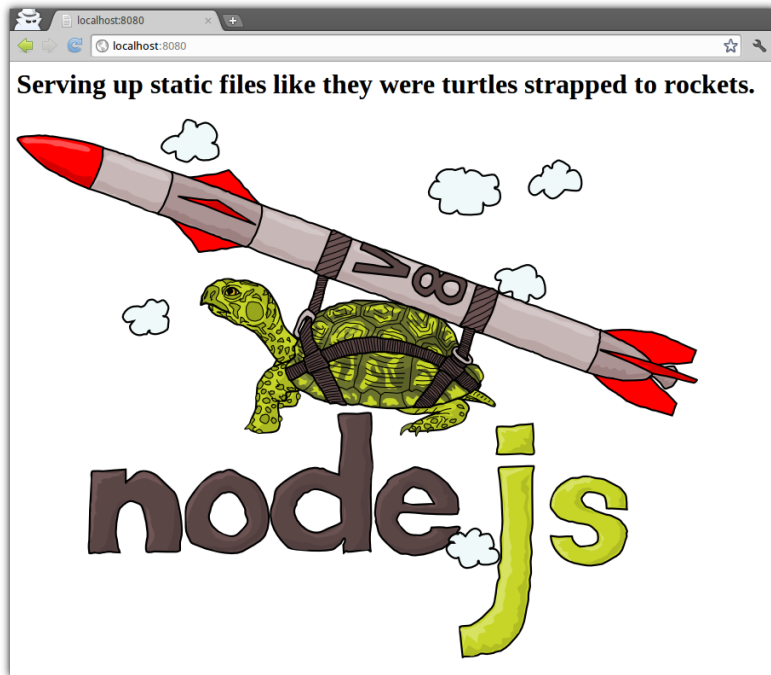
Webpack dev server



Browsersync



# http-server

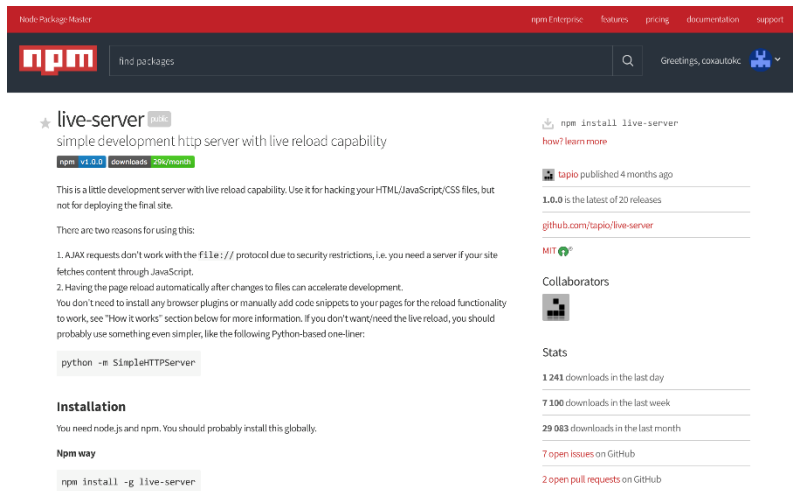


Ultra-simple

Single command serves current directory



# live-server



The screenshot shows the npm package page for 'live-server'. The header is red with the npm logo and navigation links. The main content area is white. The package name 'live-server' is at the top, followed by its description: 'simple development http server with live reload capability'. Below this, there's a section for 'This is a little development server with live reload capability. Use it for hacking your HTML/JavaScript/CSS files, but not for deploying the final site.' and a list of reasons for using it. The 'Installation' section follows, with a note about needing node.js and npm, and a code block for installing globally. The right sidebar shows the command 'npm install live-server', the version '1.0.0', the publisher 'tapio', and a list of stats including downloads and open issues.

Node Package Manager

npm Enterprise features pricing documentation support

find packages

live-server link

simple development http server with live reload capability

npm v1.0.0 downloads 29k/month

This is a little development server with live reload capability. Use it for hacking your HTML/JavaScript/CSS files, but not for deploying the final site.

There are two reasons for using this:

1. AJAX requests don't work with the `file:///` protocol due to security restrictions, i.e. you need a server if your site fetches content through JavaScript.
2. Having the page reload automatically after changes to files can accelerate development.

You don't need to install any browser plugins or manually add code snippets to your pages for the reload functionality to work, see "How it works" section below for more information. If you don't want/need the live reload, you should probably use something even simpler, like the following Python-based one-liner:

```
python -m SimpleHTTPServer
```

**Installation**

You need node.js and npm. You should probably install this globally.

**Npm way**

```
npm install -g live-server
```

npm install live-server

how? [learn more](#)

tapio published 4 months ago

1.0.0 is the latest of 20 releases

[github.com/tapio/live-server](#)

MIT

**Collaborators**

**Stats**

- 1241 downloads in the last day
- 7100 downloads in the last week
- 29083 downloads in the last month
- 7 open issues on GitHub
- 2 open pull requests on GitHub

Lightweight  
Support live-reloading



# Express

express



Express alternatives

Comprehensive

Highly Configurable

Production grade

Can run it everywhere



budo

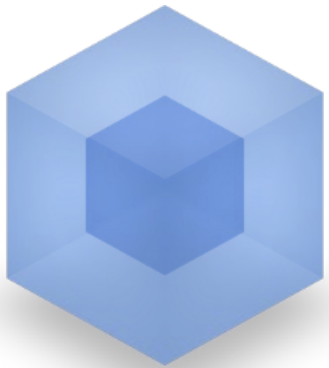
**budō**

Integrates with Browserify

Includes hot reloading



# Webpack Dev Server



**webpack**  
MODULE BUNDLER

**Built in to Webpack**

**Serves from memory**

**Includes hot reloading**





# Browsersync

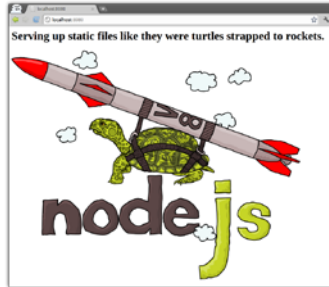


**Browsersync**

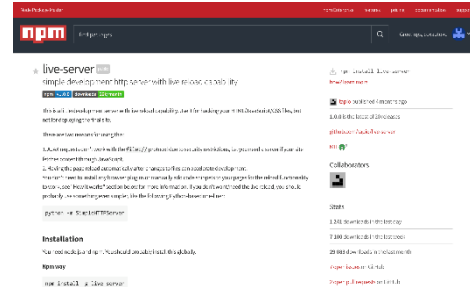
- Dedicated IP for sharing work on LAN
- All interactions remain in sync!
- Great for cross-device testing
- Integrates with Webpack, Browserify, Gulp



# Development Webservers



http-server



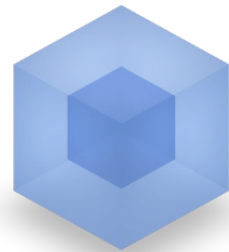
live-server

express

Express

budō

budo



webpack  
MODULE BUNDLER

Webpack dev server



Browsersync



# Demo



## Set up Development Webserver



# Sharing Work-in-progress

---



Uh, why wouldn't I just put my work on AWS, Azure, etc?

- You, my curious viewer



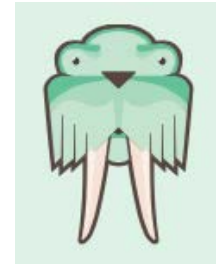
# Sharing Work-in-progress



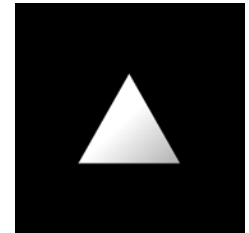
localtunnel

ngrok

ngrok



Surge



now



# localtunnel



Easily share work on your local machine

Setup:

1. `npm install localtunnel -g`
2. Start your app
3. `lt --port 3000`



# API

---

The localtunnel client is also usable through an API (for test integration, automation, etc)

## localtunnel(port [,opts], fn)

Creates a new localtunnel to the specified local `port` . `fn` will be called once you have been assigned a public localtunnel url. `opts` can be used to request a specific `subdomain` .

```
var localtunnel = require('localtunnel');

var tunnel = localtunnel(port, function(err, tunnel) {
  if (err) ...

  // the assigned public url for your tunnel
  // i.e. https://abcdefgjhij.localtunnel.me
  tunnel.url;
});

tunnel.on('close', function() {
  // tunnels are closed
});
```



# ngrok

## ngrok

Secure tunnel to your local machine

1. Sign up
2. Install ngrok
3. Install authtoken
4. Start your app
5. `./ngrok http 80`



1. Start by [downloading ngrok](#).

2. Install your authtoken

```
./ngrok authtoken BkGfqcTs2cwAfyy9NTtc_NeM8pWRzARSWUkcvL7Kd
```

3. Create your first secure tunnel

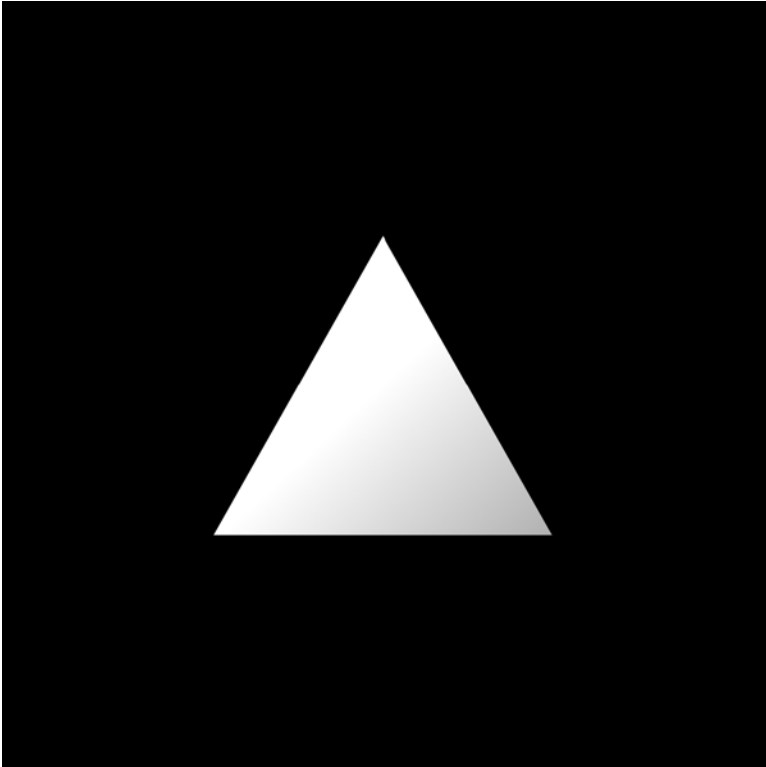
```
./ngrok http 80
```

4. Open the web interface at <http://localhost:4040> to inspect and replay requests

5. Read the [documentation](#) for instructions on advanced features like adding HTTP authentication, setting custom subdomains and more.



now



Quickly deploy Node.js to the cloud

1. `npm install -g now`
2. Create start script
3. `now`



# Surge



**Quickly host static files to public URL**

**Setup:**

**1. `npm install -g surge`**

**2. `surge`**

# Sharing Work-in-progress



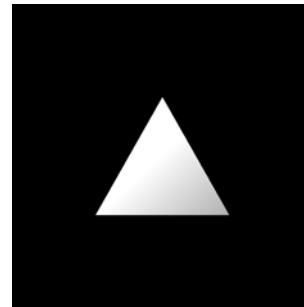
localtunnel

Easiest setup  
Ultra-versatile



ngrok

Easy setup  
Secure



now

No firewall hole  
Hosting persists



Surge

No firewall hole  
Hosting persists



# Demo



**Share Work-in-progress via localtunnel**



## Wrap Up



### Development Webservers

- http-server
- live-server
- Express
- budo
- Webpack dev server
- Browsersync

### Ways to share work-in-progress

- localtunnel, ngrok, now, surge

**Next up: Automation**

