DevServer

webpack-dev-server can be used to quickly develop an application. See the "How to Develop?" to get started.

This page describes the options that affect the behavior of webpack-dev-server (short: dev-server).

> *Options that are compatible with webpack-dev-middleware have 🔑 next to them.*

`object`

This set of options is picked up by webpack-dev-server and can be used to change its behavior in various ways. Here's a simple example that gzips and serves everything from our `dist/` directory:

```
devServer: {
  contentBase: path.join(__dirname, "dist"),
  compress: true,
  port: 9000
}
```

When the server is started, there will be a message prior to the list of resolved modules:

```
http://localhost:9000/
webpack output is served from /build/
Content not from webpack is served from /path/to/dist/
```

that will give some background on where the server is located and what it's serving.

If you're using dev-server through the Node.js API, the options in

`devServer` will be ignored. Pass the options as a second parameter instead: `new WebpackDevServer(compiler, {...})`. See here for an example of how to use webpack-dev-server through the Node.js API.

> *Be aware that when exporting multiple configurations only the `devServer` options for the first configuration will be taken into account and used for all the configurations in the array.*

> *If you're having trouble, navigating to the `/webpack-dev-server` route will show where files are served. For example, `http://localhost:9000/webpack-dev-server`.*

devServer.after

`function`

Provides the ability to execute custom middleware after all other middleware internally within the server.

```
after(app){

}
```

devServer.allowedHosts

`array`

This option allows you to whitelist services that are allowed to access the dev server.

```
allowedHosts: [
  'host.com',
  'subdomain.host.com',
  'subdomain2.host.com',
  'host2.com'
]
```

Mimicking django's `ALLOWED_HOSTS`, a value beginning with `.` can be used as a subdomain wildcard. `.host.com` will match `host.com`, `www.host.com`, and any other subdomain of `host.com`.

```
allowedHosts: [
    '.host.com',
    'host2.com'
]
```

To use this option with the CLI pass the `--allowed-hosts` option a comma-delimited string.

```
webpack-dev-server --entry /entry/file --output-path /output/path --allowed
```

devServer.before

`function`

Provides the ability to execute custom middleware prior to all other middleware internally within the server. This could be used to define custom handlers, for example:

```
before(app){
  app.get('/some/path', function(req, res) {
    res.json({ custom: 'response' });
  });
}
```

devServer.bonjour

This option broadcasts the server via ZeroConf networking on start

```
bonjour: true
```

Usage via the CLI

```
webpack-dev-server --bonjour
```

`devServer.clientLogLevel`

`string`

When using *inline mode*, the console in your DevTools will show you messages e.g. before reloading, before an error or when Hot Module Replacement is enabled. This may be too verbose.

You can prevent all these messages from showing, by using this option:

```
clientLogLevel: "none"
```

Usage via the CLI

```
webpack-dev-server --client-log-level none
```

Possible values are `none`, `error`, `warning` or `info` (default).

`devServer.color` - CLI only

`boolean`

Enables/Disables colors on the console.

```
webpack-dev-server --color
```

devServer.compress

boolean

Enable gzip compression for everything served:

```
compress: true
```

Usage via the CLI

```
webpack-dev-server --compress
```

devServer.contentBase

boolean string array

Tell the server where to serve content from. This is only necessary if you want to serve static files. devServer.publicPath will be used to determine where the bundles should be served from, and takes precedence.

By default it will use your current working directory to serve content, but you can modify this to another directory:

```
contentBase: path.join(__dirname, "public")
```

Note that it is recommended to use an absolute path.

It is also possible to serve from multiple directories:

```
contentBase: [path.join(__dirname, "public"), path.join(__dirname, "assets"
```

To disable `contentBase`:

```
contentBase: false
```

Usage via the CLI

```
webpack-dev-server --content-base /path/to/content/dir
```

`devServer.disableHostCheck`

`boolean`

When set to true this option bypasses host checking. THIS IS NOT RECOMMENDED as apps that do not check the host are vulnerable to DNS rebinding attacks.

```
disableHostCheck: true
```

Usage via the CLI

```
webpack-dev-server --disable-host-check
```

`devServer.filename` 🔑

`string`

This option lets you reduce the compilations in **lazy mode**. By default in **lazy mode**, every request results in a new compilation. With `filename`, it's possible to only compile when a certain file is requested.

If `output.filename` is set to `bundle.js` and `filename` is used like this:

```
lazy: true,
filename: "bundle.js"
```

It will now only compile the bundle when `/bundle.js` is requested.

> *`filename` has no effect when used without **lazy mode**.*

`devServer.headers` 🔑

`object`

Adds headers to all responses:

```
headers: {
  "X-Custom-Foo": "bar"
}
```

`devServer.historyApiFallback`

`boolean object`

When using the HTML5 History API, the `index.html` page will likely have to be served in place of any `404` responses. Enable this by passing:

```
historyApiFallback: true
```

By passing an object this behavior can be controlled further using options like `rewrites`:

```
historyApiFallback: {
  rewrites: [
    { from: /^\/$/, to: '/views/landing.html' },
```

```
    { from: /^\/subpage/, to: '/views/subpage.html' },
    { from: /./, to: '/views/404.html' }
  ]
}
```

When using dots in your path (common with Angular), you may need to use the `disableDotRule`:

```
historyApiFallback: {
  disableDotRule: true
}
```

Usage via the CLI

```
webpack-dev-server --history-api-fallback
```

For more options and information, see the connect-history-api-fallback documentation.

`devServer.host`

`string`

Specify a host to use. By default this is `localhost`. If you want your server to be accessible externally, specify it like this:

```
host: "0.0.0.0"
```

Usage via the CLI

```
webpack-dev-server --host 0.0.0.0
```

`devServer.hot`

`boolean`

Enable webpack's Hot Module Replacement feature:

```
hot: true
```

> *Note that `webpack.HotModuleReplacementPlugin` is required to fully enable HMR. If `webpack` or `webpack-dev-server` are launched with the `--hot` option, this plugin will be added automatically, so you may not need to add this to your `webpack.config.js`. See the [HMR concepts page](#) for more information.*

`devServer.hotOnly`

`boolean`

Enables Hot Module Replacement (see `devServer.hot`) without page refresh as fallback in case of build failures.

```
hotOnly: true
```

Usage via the CLI

```
webpack-dev-server --hot-only
```

`devServer.https`

`boolean object`

By default dev-server will be served over HTTP. It can optionally be served

over HTTP/2 with HTTPS:

```
https: true
```

With the above setting a self-signed certificate is used, but you can provide your own:

```
https: {
  key: fs.readFileSync("/path/to/server.key"),
  cert: fs.readFileSync("/path/to/server.crt"),
  ca: fs.readFileSync("/path/to/ca.pem"),
}
```

This object is passed straight to Node.js HTTPS module, so see the HTTPS documentation for more information.

Usage via the CLI

```
webpack-dev-server --https
```

To pass your own certificate via the CLI use the following options

```
webpack-dev-server --https --key /path/to/server.key --cert /path/to/server
```

devServer.index

```
string
```

The filename that is considered the index file.

```
index: 'index.htm'
```

`devServer.info` - CLI only

`boolean`

Output cli information. It is enabled by default.

```
webpack-dev-server --info=false
```

`devServer.inline`

`boolean`

Toggle between the dev-server's two different modes. By default the application will be served with *inline mode* enabled. This means that a script will be inserted in your bundle to take care of live reloading, and build messages will appear in the browser console.

It is also possible to use **iframe mode**, which uses an `<iframe>` under a notification bar with messages about the build. To switch to **iframe mode**:

```
inline: false
```

Usage via the CLI

```
webpack-dev-server --inline=false
```

> *Inline mode is recommended for Hot Module Replacement as it includes an HMR trigger from the websocket. Polling mode can be used as an alternative, but requires an additional entry point,* `'webpack/hot/poll?1000'`.

`devServer.lazy` 🔑

`boolean`

When `lazy` is enabled, the dev-server will only compile the bundle when it gets requested. This means that webpack will not watch any file changes. We call this **lazy mode**.

```
lazy: true
```

Usage via the CLI

```
webpack-dev-server --lazy
```

> *`watchOptions` will have no effect when used with **lazy mode**.*

> *If you use the CLI, make sure **inline mode** is disabled.*

`devServer.noInfo` 🔑

`boolean`

With `noInfo` enabled, messages like the webpack bundle information that is shown when starting up and after each save, will be hidden. Errors and warnings will still be shown.

```
noInfo: true
```

`devServer.open`

`boolean`

When `open` is enabled, the dev server will open the browser.

```
open: true
```

Usage via the CLI

```
webpack-dev-server --open
```

If no browser is provided (as shown above), your default browser will be used. To specify a different browser, just pass its name:

```
webpack-dev-server --open 'Google Chrome'
```

devServer.openPage

`string`

Specify a page to navigate to when opening the browser.

```
openPage: '/different/page'
```

Usage via the CLI

```
webpack-dev-server --open-page "/different/page"
```

devServer.overlay

`boolean` `object`

Shows a full-screen overlay in the browser when there are compiler errors or warnings. Disabled by default. If you want to show only compiler errors:

```
overlay: true
```

If you want to show warnings as well as errors:

```
overlay: {
  warnings: true,
  errors: true
}
```

devServer.pfx

`string`

When used via the CLI, a path to an SSL .pfx file. If used in options, it should be the bytestream of the .pfx file.

```
pfx: '/path/to/file.pfx'
```

Usage via the CLI

```
webpack-dev-server --pfx /path/to/file.pfx
```

devServer.pfxPassphrase

`string`

The passphrase to a SSL PFX file.

```
pfxPassphrase: 'passphrase'
```

Usage via the CLI

```
webpack-dev-server --pfx-passphrase passphrase
```

devServer.port

`number`

Specify a port number to listen for requests on:

```
port: 8080
```

Usage via the CLI

```
webpack-dev-server --port 8080
```

devServer.proxy

`object`

Proxying some URLs can be useful when you have a separate API backend development server and you want to send API requests on the same domain.

The dev-server makes use of the powerful http-proxy-middleware package. Checkout its documentation for more advanced usages.

With a backend on `localhost:3000`, you can use this to enable proxying:

```
proxy: {
  "/api": "http://localhost:3000"
}
```

A request to `/api/users` will now proxy the request to

`http://localhost:3000/api/users`.

If you don't want `/api` to be passed along, we need to rewrite the path:

```
proxy: {
  "/api": {
    target: "http://localhost:3000",
    pathRewrite: {"^/api" : ""}
  }
}
```

A backend server running on HTTPS with an invalid certificate will not be accepted by default. If you want to, modify your config like this:

```
proxy: {
  "/api": {
    target: "https://other-server.example.com",
    secure: false
  }
}
```

Sometimes you don't want to proxy everything. It is possible to bypass the proxy based on the return value of a function.

In the function you get access to the request, response and proxy options. It must return either `false` or a path that will be served instead of continuing to proxy the request.

E.g. for a browser request, you want to serve a HTML page, but for an API request you want to proxy it. You could do something like this:

```
proxy: {
  "/api": {
    target: "http://localhost:3000",
    bypass: function(req, res, proxyOptions) {
      if (req.headers.accept.indexOf("html") !== -1) {
```

```
        console.log("Skipping proxy for browser request.");
        return "/index.html";
      }
    }
  }
}
```

If you want to proxy multiple, specific paths to the same target, you can use an array of one or more objects with a `context` property:

```
proxy: [{
  context: ["/auth", "/api"],
  target: "http://localhost:3000",
}]
```

`devServer.progress` - CLI only

`boolean`

Output running progress to console.

```
webpack-dev-server --progress
```

`devServer.public`

`string`

When using *inline mode* and you're proxying dev-server, the inline client script does not always know where to connect to. It will try to guess the URL of the server based on `window.location`, but if that fails you'll need to use this.

For example, the dev-server is proxied by nginx, and available on `myapp.test`:

```
public: "myapp.test:80"
```

Usage via the CLI

```
webpack-dev-server --public myapp.test:80
```

`devServer.publicPath` 🔑

`string`

The bundled files will be available in the browser under this path.

Imagine that the server is running under `http://localhost:8080` and `output.filename` is set to `bundle.js`. By default the `publicPath` is `"/"`, so your bundle is available as `http://localhost:8080/bundle.js`.

The `publicPath` can be changed so the bundle is put in a directory:

```
publicPath: "/assets/"
```

The bundle will now be available as `http://localhost:8080/assets/bundle.js`.

> *Make sure `publicPath` always starts and ends with a forward slash.*

It is also possible to use a full URL. This is necessary for Hot Module Replacement.

```
publicPath: "http://localhost:8080/assets/"
```

The bundle will also be available as `http://localhost:8080/assets/bundle.js`.

> *It is recommended that `devServer.publicPath` is the same as `output.publicPath`.*

`devServer.quiet` 🔑

`boolean`

With `quiet` enabled, nothing except the initial startup information will be written to the console. This also means that errors or warnings from webpack are not visible.

```
quiet: true
```

Usage via the CLI

```
webpack-dev-server --quiet
```

`devServer.setup`

`function`

> *This option is **deprecated** in favor of `before` and will be removed in v3.0.0.*

Here you can access the Express app object and add your own custom middleware to it. For example, to define custom handlers for some paths:

```
setup(app){
  app.get('/some/path', function(req, res) {
    res.json({ custom: 'response' });
  });
}
```

devServer.socket

`string`

The Unix socket to listen to (instead of a host).

```
socket: 'socket'
```

Usage via the CLI

```
webpack-dev-server --socket socket
```

devServer.staticOptions

It is possible to configure advanced options for serving static files from `contentBase`. See the Express documentation for the possible options. An example:

```
staticOptions: {
  redirect: false
}
```

> *This only works when using `contentBase` as a `string`.*

devServer.stats 🔑

`string` `object`

This option lets you precisely control what bundle information gets displayed. This can be a nice middle ground if you want some bundle information, but not all of it.

To show only errors in your bundle:

```
stats: "errors-only"
```

For more information, see the **stats documentation**.

> *This option has no effect when used with `quiet` or `noInfo`.*

`devServer.stdin` - CLI only

`boolean`

This option closes the server when stdin ends.

```
webpack-dev-server --stdin
```

`devServer.useLocalIp`

`boolean`

This option lets the browser open with your local IP.

```
useLocalIp: true
```

Usage via the CLI

```
webpack-dev-server --useLocalIp
```

`devServer.watchContentBase`

`boolean`

Tell the server to watch the files served by the `devServer.contentBase` option. File changes will trigger a full page reload.

```
watchContentBase: true
```

It is disabled by default.

Usage via the CLI

```
webpack-dev-server --watch-content-base
```

`devServer.watchOptions` 🔑

`object`

Control options related to watching the files.

webpack uses the file system to get notified of file changes. In some cases this does not work. For example, when using Network File System (NFS). Vagrant also has a lot of problems with this. In these cases, use polling:

```
watchOptions: {
  poll: true
}
```

If this is too heavy on the file system, you can change this to an integer to set the interval in milliseconds.

See WatchOptions for more options.