Hot Reload

The highest impact on your application's bootstrapping process is **TypeScript compilation**. Fortunately, with webpack HMR (Hot-Module Replacement), we don't need to recompile the entire project each time a change occurs. This significantly decreases the amount of time necessary to instantiate your application, and makes iterative development a lot easier.

> warning **Warning** Note that `webpack` won't automatically copy your assets (e.g. `graphql` files) to the `dist` folder. Similarly, `webpack` is not compatible with glob static paths (e.g., the `entities` property in `TypeOrmModule`).

With CLI

If you are using the Nest CLI, the configuration process is pretty straightforward. The CLI wraps `webpack`, which allows use of the `HotModuleReplacementPlugin`.

Installation

First install the required packages:

```
$ npm i --save-dev webpack-node-externals run-script-webpack-plugin
webpack
```

> info **Hint** If you use **Yarn Berry** (not classic Yarn), install the `webpack-pnp-externals` package instead of the `webpack-node-externals`.

Configuration

Once the installation is complete, create a `webpack-hmr.config.js` file in the root directory of your application.

```
const nodeExternals = require('webpack-node-externals');
const { RunScriptWebpackPlugin } = require('run-script-webpack-plugin');

module.exports = function (options, webpack) {
  return {
    ...options,
    entry: ['webpack/hot/poll?100', options.entry],
    externals: [
      nodeExternals({
        allowlist: ['webpack/hot/poll?100'],
      }),
    ],
    plugins: [
      ...options.plugins,
      new webpack.HotModuleReplacementPlugin(),
      new webpack.WatchIgnorePlugin({
        paths: [/\.js$/, /\.d\.ts$/],
      }),
      new RunScriptWebpackPlugin({ name: options.output.filename,
autoRestart: false }),
    ],
  };
};
```

> info **Hint** With **Yarn Berry** (not classic Yarn), instead of using the `nodeExternals` in the `externals` configuration property, use the `WebpackPnpExternals` from `webpack-pnp-externals` package: `WebpackPnpExternals({{ '{' }} exclude: ['webpack/hot/poll?100'] {{ '}' }})`.

This function takes the original object containing the default webpack configuration as a first argument, and the reference to the underlying `webpack` package used by the Nest CLI as the second one. Also, it returns a modified webpack configuration with the `HotModuleReplacementPlugin`, `WatchIgnorePlugin`, and `RunScriptWebpackPlugin` plugins.

Hot-Module Replacement

To enable **HMR**, open the application entry file (`main.ts`) and add the following webpack-related instructions:

```
declare const module: any;

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);

  if (module.hot) {
    module.hot.accept();
    module.hot.dispose(() => app.close());
  }
}
bootstrap();
```

To simplify the execution process, add a script to your `package.json` file.

```
"start:dev": "nest build --webpack --webpackPath webpack-hmr.config.js --
watch"
```

Now simply open your command line and run the following command:

```
$ npm run start:dev
```

Without CLI

If you are not using the Nest CLI, the configuration will be slightly more complex (will require more manual steps).

Installation

First install the required packages:

```
$ npm i --save-dev webpack webpack-cli webpack-node-externals ts-loader
run-script-webpack-plugin
```

> info **Hint** If you use **Yarn Berry** (not classic Yarn), install the `webpack-pnp-externals` package instead of the `webpack-node-externals`.

Configuration

Once the installation is complete, create a `webpack.config.js` file in the root directory of your application.

```
const webpack = require('webpack');
const path = require('path');
const nodeExternals = require('webpack-node-externals');
const { RunScriptWebpackPlugin } = require('run-script-webpack-plugin');

module.exports = {
  entry: ['webpack/hot/poll?100', './src/main.ts'],
  target: 'node',
  externals: [
    nodeExternals({
      allowlist: ['webpack/hot/poll?100'],
    }),
  ],
  module: {
    rules: [
      {
        test: /.tsx?$/,
        use: 'ts-loader',
        exclude: /node_modules/,
      },
    ],
  },
  mode: 'development',
  resolve: {
    extensions: ['.tsx', '.ts', '.js'],
  },
  plugins: [
    new webpack.HotModuleReplacementPlugin(),
    new RunScriptWebpackPlugin({ name: 'server.js', autoRestart: false }),
  ],
  output: {
    path: path.join(__dirname, 'dist'),
    filename: 'server.js',
  },
};
```

> info **Hint** With **Yarn Berry** (not classic Yarn), instead of using the `nodeExternals` in the `externals` configuration property, use the `WebpackPnpExternals` from `webpack-pnp-externals` package: `WebpackPnpExternals({{ '{' }} exclude: ['webpack/hot/poll?100'] {{ '}' }})`.

This configuration tells webpack a few essential things about your application: location of the entry file, which directory should be used to hold **compiled** files, and what kind of loader we want to use to compile source files. Generally, you should be able to use this file as-is, even if you don't fully understand all of the options.

Hot-Module Replacement

To enable **HMR**, open the application entry file (`main.ts`) and add the following webpack-related instructions:

```
declare const module: any;

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(3000);

  if (module.hot) {
    module.hot.accept();
    module.hot.dispose(() => app.close());
  }
}
bootstrap();
```

To simplify the execution process, add a script to your `package.json` file.

```
"start:dev": "webpack --config webpack.config.js --watch"
```

Now simply open your command line and run the following command:

```
$ npm run start:dev
```

Example

A working example is available here.