

# BUILD & SETUP

Ori Calvo, 2017

[oric@trainologic.com](mailto:oric@trainologic.com)

<http://trainologic.com>

# Keep Maintaining the Magic

2

- A new developer that just joined the team should be able to
  - ▣ git clone
  - ▣ npm install
  - ▣ npm start
- That's it !!!
- All servers are up and running
- Browsers is opened pointing to the correct URL
- Exploration now begins ...

# Build

3

- A real life production project requires the following
  - Typescript compilation
  - SASS compilation
  - Module loader
  - Dev web server
  - Bundling & minification
  - Optimization like AOT & Tree shaking
  - Localization (build or runtime)

# Build changes a lot

4

- The build process is just a reflection of the technology stack we are using
- Agile technology stack → Agile build
- Most popular IDEs are not capable of handling this level of agility
- This is the main reason for the rise of task/build runners such as Grunt/Gulp/Webpack

# NodeJS as Build tool

5

- NodeJS has an extreme eco-system and therefore is most suited for implementing the build scripts
- Once build is IDE-independent a single development team may work with multiple types of IDEs/platforms
- Still, you need to think about the level of integration between external build and your preferred IDE

# NPM

6

- ❑ The largest ecosystem of open source in the history
- ❑ NPM stands for Node Package Manager
- ❑ Packages represent a collection of code
- ❑ NPM automates installation and updating of packages according to dependencies
- ❑ Dependency is simply a code that relies on another code in order to function

# SEMVER

7

- SEMVER stands for semantic versioning
  - ▣ Meaning version is not just a number
  - ▣ It has semantic
- SEMVER helps other developers to watch for new features or any 'breaking change'
- 2.1.0 → MAJOR.MINOR.PATCH
  - ▣ Minor and patch should not contain breaking changes
  - ▣ Major might break your code

# npm install [PACKAGE]

8

- Installs a new package to **node\_modules**
- Adds it to the **dependencies** section inside **package.json**
  - ▣ Version being used is ^2.1.0
- Installs all dependencies recursively
- Starting NPM 5 **package-lock.json** is updated automatically



# devDependencies

9

- Some packages are required only during development
  - ▣ Typescript
  - ▣ SASS
- Use **npm install --save-dev**
- Running “npm install” brings both dependencies and devDependencies
- If `NODE_ENV=production`
  - ▣ Does not install devDependencies

# Conflict Resolution

10

- ❑ Two different packages (A,B) may have same dependency (C)
- ❑ NPM prefer to “push” C up the directory structure so it can be shared by both
- ❑ In case of A & B needs different versions of C NPM keep a copy of C under B
- ❑ Two different versions of C might be loaded at runtime 😞

# peerDependency

11

- C is peerDependency of A
- When installing A, C is not installed
  - ▣ You get an error instead
- Now, it's the app responsibility to install the peer dependency
- Using peerDependency allows different plugins
  - ▣ To have no local copy of the host
  - ▣ Therefore can require the same host

# npm outdated

12

- Get a list of all packages that are outdated
- This command is usually followed by “npm update”
  - ▣ Which installs latest version of all packages
- You should note that SEMVER limitation are effective
- If package.json says ^1.0.0 version 2.0.0 will not be installed

# package-lock.json

13


- Having a dependency of type `^1.0.0` means that two different installations might end with different version of the dependency
- `package-lock.json` solves that by recursively lists all dependencies and their exact version
- No need any more for **npm shrinkwrap**

# NPM Scripts – as a build tool, example

14

- ❑ 'npm install node-sass' command will install a package which will compile sass to css

```
{
  "name": "scripts",
  "version": "1.0.0",
  "description": "",
  "main": "hello.js",
  "scripts": {
    "build": "node-sass --output-style compressed -o ./cssfolder ./sassfolder",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```



# npm link

15

- Assuming you are developing a library
- You want to test it inside an application
- However the library is not published yet
- Use **npm link** inside the library
- Use **npm link [LIB]** inside the application
- A symbolic link is created inside the host application's `node_modules` folders

# Publish an npm package

16

- Run **npm init**
  - ▣ package.json is created
- Create a new user using npm web site
- Authenticate using **npm login**
- Run **npm publish**
- Consider **npm version patch** to increase version number before publishing



# yarn

17

- ❑ Yarn is a package manager created by Facebook
- ❑ For most cases **yarn** follows the same rules and configuration as npm
- ❑ Is considered having better caching strategy and generally is faster
- ❑ Starting NPM 5 the performance benefit still exists but is lower

# Yarn --offline

18

- Got the package.json file / yarn.lock but for some reason internet connectivity is not available?
- **yarn --offline** will install all of the package.json file dependencies even without internet

# @angular/cli

19

- ❑ Even when using **Webpack**, implementing build scripts is considered a complex task
- ❑ So the Angular team created an abstraction layer on top of Webpack
  - ▣ So now you need to learn both ...
- ❑ Starting with @angular/cli is easy
- ❑ At the long term you understand that customization capabilities resides inside Webpack and not inside angular/cli

# Weback

20

- A module loader/bundler
  - ▣ Even for development purposes
- Tries to bundle everything
  - ▣ CSS
  - ▣ HTML
  - ▣ Images
- Extremely configurable
- An impressive eco-system

# Webpack core concepts

21

- **Entry** – The starting point of the module graph
- **Output** – The resultant bundle
- **Loader**
  - ▣ Webpack only understands JavaScript
  - ▣ Loaders allows Webpack to handle non JavaScript files
  - ▣ Are focused around transformation
- **Plugin**
  - ▣ Handles anything except module transformation

# webpack.config.js

22

- ❑ Webpack uses JavaScript (not JSON) to describe configuration
- ❑ It allows us to create dynamic configuration

```
module.exports = {  
  entry: './app/main.ts',  
  
  output: {  
    path: path.resolve(__dirname, 'dist'),  
    filename: 'bundle.js',  
  },  
  
  resolve: {  
    extensions: ['.ts', '.js'],  
  }  
}
```

Can even return a promise so configuration can be created using async operations

Inside Typescript files we are not using any extension

# webpack.config.js

23

- Although Webpack is very sophisticated it is still not able to support Angular completely
- @ngtools/webpack knows how to fill the gap

```
module.exports = {  
  module: {  
    rules: [  
      {  
        test: /\.ts$/,  
        loader: '@ngtools/webpack',  
      },  
    ],  
  },  
  plugins: [  
    new AngularCompilerPlugin ({  
      tsConfigPath: 'tsconfig.json',  
      mainPath: 'main.ts',  
      skipCodeGeneration: true,  
    }),  
  ],  
}
```

Must configure  
both a loader and  
a plugin

Don't miss that !!!  
Without that,  
@ngtools outputs  
AOT classes

# @ngtools/webpack

24

- ❑ Handles Typescript compilation
- ❑ templateUrl → template
- ❑ styleUrls → styles
- ❑ Detects lazy loaded modules and creates a bundle for each one
- ❑ Supports AOT



# Webpack Dev Server

25

- A NodeJS application that hosts Webpack
- Blocks HTTP requests until build is completed
- Supports live reload
- Generates bundles in memory and serves them through HTTP
- How to
  - ▣ `yarn add webpack-dev-server`
  - ▣ `node_modules/.bin/webpack-dev-server`

# Fixing index.html

26

- Webpack generates bundles
- Not all bundles are known statically
  - ▣ Think lazy loading
- You need a way to fix index.html with all generated bundles
- Use **HtmlWebpackPlugin**
- **yarn add html-webpack-plugin**

# HtmlWebpackPlugin

27

Template is  
relative to  
webpack.config.js

```
plugins: [  
  new HtmlWebpackPlugin({  
    title: 'My App',  
    template: './index.html',  
    filename: './index.html'  
  })  
]
```

filename is  
relative to  
output.path

```
<!DOCTYPE html>  
<html lang="en">  
<head>  
  <meta charset="UTF-8">  
  <title>Title 2</title>  
  
  <script src="node_modules/reflect-metadata/Reflect.js"></script>  
  <script src="node_modules/zone.js/dist/zone.js"></script>  
</head>  
<body>  
  <my-app></my-app>  
  <script type="text/javascript" src="bundle.js"></script>  
</body>  
</html>
```

This is the  
generated  
index.html

# template → templateUrl

28

- Up until now we used inline template
- Moving to **templateUrl** is a bit challenging since Angular does not support relative URL ☹️

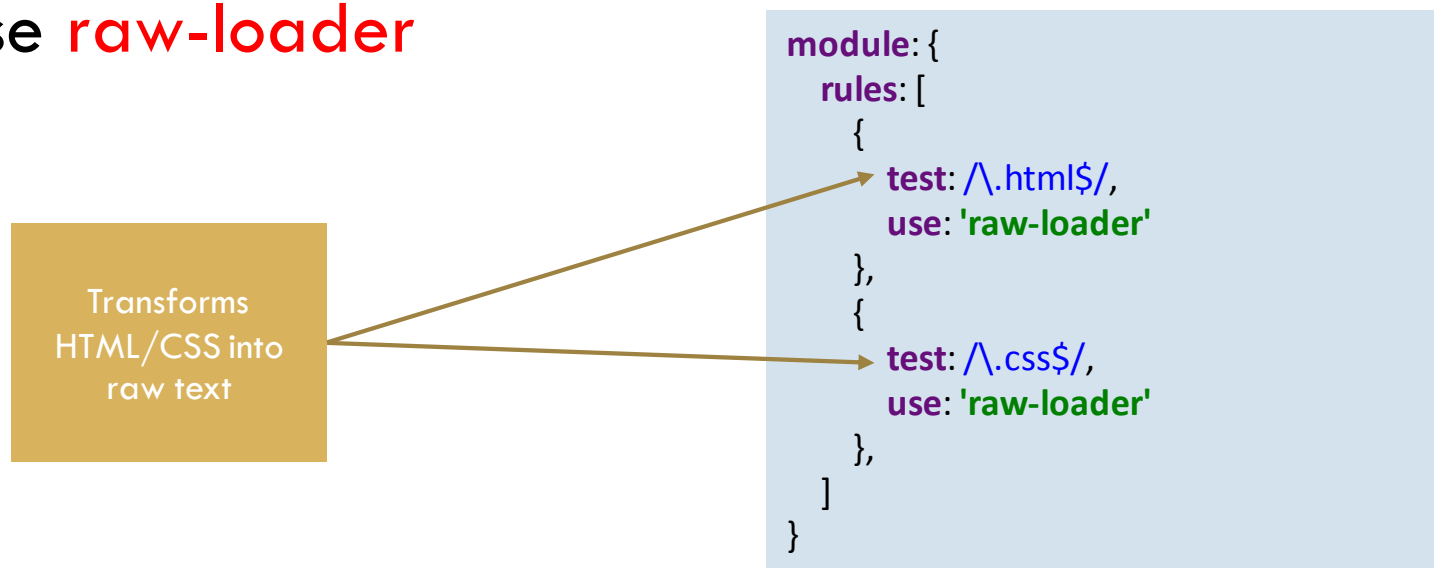
```
@Component({  
  selector: "my-app",  
  templateUrl: "app.component.html",  
})  
export class AppComponent {  
}
```

- Angular looks for **app.component.html** at the root path and not next to **app.component.ts**

# HTML & CSS Loader

29

- @ngtools is able to transform **template** syntax to **templateUrl**
- However it does not know how to load the HTML
- Use **raw-loader**



# Handling CSS

30

- CSS may contain `@import` and `url(...)`
- The raw-loader returns the CSS as is
- The bundle does not contain those assets → Runtime error
- Use **css-loader**
- Transforms `url(..)` into plain `require`
- `url("image.jpg")` → `require("../image.jpg")`
- Now you need another loader for handling JPG
  - ▣ **file-loader**
  - ▣ **url-loader**

# Handling CSS

31

Order matter  
First loader runs  
last

```
module: {  
  rules: [  
    {  
      test: /\.css$/,  
      use: [  
        // "exports-loader?module.exports.toString()",  
        "to-string-loader",  
        'css-loader',  
      ],  
    },  
    {  
      test: /\.jpg$/,  
      loader: 'url-loader',  
    },  
  ]  
}
```

# @angular/cli Getting Started

32

- Install CLI tool globally: `yarn global add @angular/cli`
- Verify installation: `ng version`
- Generate new project: `ng new my-project`
  - ▣ A new directory is created with all source files
  - ▣ Automatically restore packages using yarn



# ng new options

33


- **--directory**: Name of directory to create, by default this is the application name
- **--prefix**: Component selector prefix
  - ▣ Can be overridden per component
- **--inline-style**: Do not generate CSS file
  - ▣ Can be overridden per component
- **--inline-template**: Do not use inline templates
  - ▣ Can be overridden per component

# --routing

34

- Commonly used cli command option to create a new project and automatically add a routing file in order to implement routing in angular app
- **ng new myapp --routing**

The project files tree after the command.  
A routing module file is now available



```
src
-app
----app.component.css
----app.component.html
----app.component.spec.ts
----app.component.ts
----app.module.ts
----app-routing.module.ts
-assets
-environments
-favicon.ico
-index.html
-polyfills.ts
-main.ts
-styles.css
-test.ts
-tsconfig.app.json
-typings.d.ts
-tsconfig.spec.json
```

# ng generate

35

- Assists in creating features to the app such as components, modules, services, pipes, directive etc
- Some options are derived from project level definition
- Some options can be re-defined
- Also have other options such as:
  - ▣ `--inline-template` use an inline template instead of a separate HTML file
  - ▣ `--inline-style` use inline styles instead of a separate CSS file
  - ▣ `--prefix` change prefix selector

# --flat

36

- Do not generate a parent directory when generating a new component
- `ng g component contactList --flat`
- Probably you will want to use it when defining a new root component per feature module
  - ▣ To be consistent with `app.component.ts`

# ng build

37

- **ng serve** starts a development server and all JavaScript bundles are created in memory
- You can only analyze the bundles using a browser !!!
- **ng build** generates bundles under **outDir** which is the **dist** folder
  - ▣ Thus you can now analyze the bundles
- Those bundles are not minified and optimized
- Use them for development purpose only

# ng build --prod

38

- Uses prod environment settings
  - ▣ See more details later
- Enable AOT
- Add hash values for all files
- No source maps
- Minification
- Extract CSS
  - ▣ Only styles.css

# target vs. environment

39

- target effects the output
  - ▣ AOT
  - ▣ Minification
  - ▣ More ...
- environment effects some global variables that can be read at runtime and change the way the application behaves

# Environments

40

- Angular/cli automatically creates **environment.ts**

```
export const environment = {  
  production: false  
};
```

- You may add additional configuration fields
- For each additional environment you should create an separate file. For example, **environment.prod.ts**

```
export const environment = {  
  production: true  
};
```



# Environments

41

- During build Angular/cli overrides environment.ts with environment.[XXX].ts
- Your code should reference environment.ts only

```
import {environment} from "../environments/environment";

export class AppComponent {
  title = 'app works!';

  constructor() {
    console.log(environment.name);
  }
}
```

May change  
according to  
active environment

ng serve --environment=prod

# Angular prodMode

42

- During development Angular creates a “debug friendly” code
- To enable optimization you must invoke

```
if (environment.production) {  
  enableProdMode();  
}
```

- At runtime you can check

```
import {Component, isDevMode} from '@angular/core';  
  
console.log("isDevMode", isDevMode());
```

# Ejecting application

43

- By default, Angular CLI manages the underlying webpack configuration for you so you don't have to deal with its complexity
- However, smart customization requires `webpack.config.js` editing
- Use **ng eject**

# ng eject

44

- A property **ejected**: true is added to .angular-cli.json
- A **webpack.config.js** file is generated
- package.json is fixed to use non ng tools

```
"scripts": {  
  "ng": "ng",  
  "start": "webpack-dev-server --port=4200",  
  "build": "webpack",  
}
```

- And additional packages are added

```
"devDependencies": {  
  "webpack-dev-server": "~2.4.2",  
  "autoprefixer": "^6.5.3",  
  "css-loader": "^0.27.3",  
}
```

# webpack.config.js

45

- ❑ More than 900 lines of code/configuration 😞
- ❑ Loaders being used

json	css
raw	post-css
file	exports
url	sass/less/stylus
@ngtools/webpack	source-map

- ❑ Plugins

Progress	HtmlWebpack
ExtractText	NoEmitOnErrorsPlugin
GlobCopyWebpackPlugin	BaseHrefWebpackPlugin
AotPlugin	CommonsChunkPlugin

# ng eject --prod

46

- After ejecting Webpack configuration you are no longer able to use ng serve/build commands
  - ▣ You can still use ng generate commands
- So what about production build ?
- The generated webpack.config.js is for development
- Consider running **ng eject --prod**
  - ▣ It is your responsibility for managing multiple Webpack configuration files

# Serving static files

47

- Development server rejects HTTP requests for static files
- Unless the URL starts with assets and the file is located under the assets directory

```
{  
  "apps": [  
    {  
      "assets": [  
        "assets",  
        "favicon.ico"  
      ]  
    }  
  ]  
}
```

# Integrating REST API server

48

- For any given URL, the development server returns the index.html
  - ▣ Thus, serving correctly client side URLs
- However, it will do that even for `/api/` requests
- We need a way to tell Webpack to “pass through” any api HTTP request to the REST server
  - ▣ proxy
  - ▣ ng build + REST server
  - ▣ REST server + web pack middleware



# Configure backend proxy

49

- Create a JSON file named **proxy.config.json**
  - ▣ Can be any name

```
{  
  "/api/*":{  
    "target":"http://localhost:3000",  
    "secure":false,  
    "logLevel":"debug"  
  }  
}
```

We assume  
backend is  
running on port  
3000

- Execute ng serve with the correct proxy config
- **ng s --proxy-config proxy.config.json**

# Proxy after ng eject

50

- Webpack dev server supports the concept of proxy

```
"devServer": {  
  "historyApiFallback": true,  
  proxy: {  
    '/api': {  
      target: 'http://localhost:3000',  
      secure: false  
    }  
  }  
}
```

- As always Webpack offers much more customization capabilities

# Merging servers

51

- In case your backend is written using NodeJS it is annoying having to maintain two servers
  - ▣ Web pack dev server – For build
  - ▣ Express server – For REST API
- Solution 1
  - ▣ Let Webpack generate bundles: `ng build`
  - ▣ Serve bundles: `app.use(express.static("dist"))`
- Solution 2
  - ▣ Webpack middleware

# Webpack-dev-middleware

52

- First, run **ng eject**
- Second, host Webpack inside the express server

```
const express = require('express');
const webpackMiddleWare = require('webpack-dev-middleware');
const webpack = require('webpack');
const webpackConfig = require('./webpack.config');

const app = express();

app.use(webpackMiddleWare(webpack(webpackConfig)));

app.listen(3000);
```

# Summary

53

- ❑ Angular/cli is an abstraction layer on top of Webpack
- ❑ As such it makes life easier (short term)
- ❑ At the long term you should eject Webpack configuration as use it directly