Developing Angular 4 with Apache

By: Lincoln Gup, U.S. Patent and Trademark Office

The Problem

To simulate some production environment REST services, I needed to serve an Angular application from Apache on my local machine. I've been enjoying using Angular CLI's "ng serve" functionality, and wanted to preserve its convenience. The "ng serve" command builds the application, starts a web server and runs the application, and automatically rebuilds the application if you change any of the application's source code files. Apache would provide the Web services, and I needed to automate the watch and build processes.

Choosing the Solution

The solution needed to work on my local machine with the following technology stack:

- Angular 4.3.4
- Apache 2.4.23
- Windows 7 Enterprise SP1

The "ng serve" command has options like `--output-path` that appear to make the command the obvious answer. Just set some command line options, set up a shell alias or batch script to do the Angular build and deposit the code in Apache, and get back to work, right? But at the bottom of the wiki documentation page for the command is a crucial Note: "the compiled output is served from memory, not from disk." The serve command doesn't generate build output, so the files wouldn't be available to deploy to Apache, and "ng serve" wouldn't work; I needed another approach.

Angular CLI's "ng build" command can compile the application, put the output into a specified directory, and rebuild the application when files are changed. We'll look at two ways to implement a build using "ng build": npm and gulp.

You'll need to up a directory in Apache for use by the Angular application. I've created a directory named `tm-pro-se` that lives in the Apache web root directory. The Apache server appends the path from a requested URL to the document root to determine the path to a document. I've added a `ServerName` setting to Apache's virtual host file, as shown below.

```
<VirtualHost *:80>
    ServerName mylaptop.com
    DocumentRoot "c:/wamp64/www"
    <Directory  "c:/wamp64/www/">
        Options +Indexes +Includes +FollowSymLinks +MultiViews
        # all .htaccess directives are allowed
        AllowOverride All
        # deny remote access
        Require local
    </Directory>
</VirtualHost>
```

This sets the hostname that the server uses to identify itself. It allows me to access the Angular application with the URL `http://mylaptop.com/tm-pro-se/`, which will set any `Origin` request headers originating from the application to the domain `mylaptop.com`.

Build with Npm

When you create a new Angular application using Angular CLI's "ng new" command, it provides a default set of source and environment files. The files include an npm package.json file stocked with a build command. You can modify the `scripts` property in the package.json file so that you can use npm to manage the Apache build process. Note that you must adhere to the JSON format.

```
"scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "test": "ng test",
    "lint": "ng lint",
    "e2e": "ng e2e",
    "apache": "ng build --output-path C:/wamp64/www/tm-pro-se/ --deploy-url tm-pro-se/
--watch"
},
```

The Apache build script is then run from the command line:

```
npm run apache
```

The `--output-path` option specifies the directory where the build files will be placed. When the application runs, the index page, `index.html`, loads six webpack-bundled JavaScript files. The `--deploy-url` pathname is prepended to those scripts to specify a relative URL address for locating the JavaScript files and any other documents or resources. The `--watch` option reruns the build when files are changed.

Build with Gulp

If there are other tasks you want to perform around moving the code into Apache and providing the watch-n-rebuild service, you'll need a task runner. To drive this, I chose Gulp. I went with Gulp because I find the Node streams easier to read than Grunt's configuration array descriptions. Gulp is also generally faster.

We'll take a look at the various pieces of the code. The full Gulp script is provided at the bottom of this paper.

You'll need to install Gulp:

```
npm install gulp --save-dev
```

The script needs to include several modules. I always set the `use strict` directive to try to catch any bad syntax.

```
"use strict";

// include gulp
var gulp = require('gulp');
var exec = require('child_process').exec;
```

The Angular application files will be placed in the `tm-pro-se` directory under the Apache web root directory.

```
/*
 * Apache destination directory for the build. This must have a trailing
 * "/" because the pathname is prepended to resource filenames by webpack.
 */
var buildDirectory = 'tm-pro-se/';
// Apache destination pathname for the build
var buildPathname = 'C:/wamp64/www/' + buildDirectory;
```

The `build:apache` task uses Angular CLI's "ng build" command to compile, move, and watch the application. The task includes two accommodations to ensure that all build and error messages are made visible. First, Node's `child_process.exec()` function limits the amount of data sent to stdout and stderr. The `exec` function buffers its output and provides it when the command has finished executing. If the buffer limit is exceeded, the child process is terminated, either prematurely ending the build or quietly dropping messages. To ensure the build completes and view all of the messages, we need to set the `maxBuffer` option.

How do you determine a good size setting for `maxBuffer`? When you run the build, the "ng build" command will produce messages like:

```
69% building modules 815/817 modules 2 active ...node_modules\rxjs\util\MapPoly
```

If you see the same amount of messages when using the Gulp script that you would when running the "ng build" command by itself, the `maxBuffer` option is set properly.

Second, we want to ensure that we capture all stdout and stderr events for output in real time. The `exec` command returns a ChildProcess object that's an EventEmitter. We can use that to listen for stdout and stderr events using the `on()` function.

The callback function, `cb()`, is used to manage concurrency, denoting the end of an asynchronous task.

```
// build the project into the Apache directory
gulp.task('build:apache', function (cb) {
    console.log("build:apache: Building into " + buildPathname);

    /*
     * maxBuffer specifies the largest amount of data allowed on stdout or stderr.
     * The default value of maxBuffer is 200KB. If this value is exceeded, then
     * the child process is killed.
     * --output-path puts the build in the specified directory
     * --deploy-url sets the pathname for the automatically included scripts
     * --watch watches for code changes, and does a rebuild
     */
    var apacheBuildProcess = exec('ng build' +
        ' --output-path ' + buildPathname +
        ' --deploy-url ' + buildDirectory +
        ' --watch',
        {maxBuffer: 1024 * 1000},
        function (error, stdout, stderr) {
            stderr && console.error(stderr);
            cb(error);
        }
    );
    // build results are reported on stdout in real time
    apacheBuildProcess.stdout.on('data', function(data) {
```

```
        process.stdout.write('apacheBuildProcess: '+ data);
    });
    // build progress is reported on stderr in real time
    apacheBuildProcess.stderr.on('data', function(data) {
        process.stdout.write(data);
    });
});
```

The `apache` task provides a shortcut for running the tasks needed for the Apache build.

```
gulp.task('apache', gulp.series('build:apache', function(done) {
    // do more stuff
    done();
}));
```

I've only specified one task, but this is format allows you to easily add other tasks to the list. For example, we could add a task to do `anotherThing` following the `build:apache` task:

```
gulp.task('apache', gulp.series('build:apache', 'anotherThing', function(done) {
    // do more stuff
    done();
}));
```

To run the build – move the code into Apache and perform the watch-n-rebuild service – simply enter the command:

```
gulp apache
```

The build is complete when you see the "ng build" bundle messages such as:

```
Time: 21573ms
chunk    {0} polyfills.bundle.js, polyfills.bundle.js.map (polyfills) 252 kB {5}
[initial] [rendered]
chunk    {1} main.bundle.js, main.bundle.js.map (main) 976 kB {4} [initial] [rendered]
chunk    {2} scripts.bundle.js, scripts.bundle.js.map (scripts) 127 kB {5} [initial]
[rendered]
chunk    {3} styles.bundle.js, styles.bundle.js.map (styles) 11.3 kB {5} [initial]
[rendered]
chunk    {4} vendor.bundle.js, vendor.bundle.js.map (vendor) 3.27 MB [initial]
[rendered]
chunk    {5} inline.bundle.js, inline.bundle.js.map (inline) 0 bytes [entry]
[rendered]
```

Complete Gulp Script

Below is the Gulp script in its entirety.

```
"use strict";

// include gulp
var gulp = require('gulp');
var exec = require('child_process').exec;

/*
 * Apache destination directory for the build. This must have a trailing
 * "/" because the pathname is prepended to resource filenames by webpack.
 */
var buildDirectory = 'tm-pro-se/';
// Apache destination pathname for the build
var buildPathname = 'C:/wamp64/www/' + buildDirectory;
```

```
// build the project into the Apache directory
gulp.task('build:apache', function (cb) {
    console.log("build:apache: Building into " + buildPathname);

    /*
     * maxBuffer specifies the largest amount of data allowed on stdout or stderr.
     * The default value of maxBuffer is 200KB. If this value is exceeded, then
     * the child process is killed.
     * --output-path puts the build in the specified directory
     * --deploy-url sets the pathname for the automatically included scripts
     * --watch watches for code changes, and does a rebuild
     */
    var apacheBuildProcess = exec('ng build' +
        ' --output-path ' + buildPathname +
        ' --deploy-url ' + buildDirectory +
        ' --watch',
        {maxBuffer: 1024 * 1000},
        function (error, stdout, stderr) {
            stderr && console.error(stderr);
            cb(error);
        }
    );
    // build results are reported on stdout in real time
    apacheBuildProcess.stdout.on('data', function(data) {
        process.stdout.write('apacheBuildProcess: '+ data);
    });
    // build progress is reported on stderr in real time
    apacheBuildProcess.stderr.on('data', function(data) {
        process.stdout.write(data);
    });
});

gulp.task('apache', gulp.series('build:apache', function(done) {
    // do more stuff
    done();
}));
```

References

Some useful links.

Gulp: https://github.com/gulpjs/gulp/tree/master/docs
Npm: https://docs.npmjs.com/
Angular CLI: https://github.com/angular/angular-cli/wiki
Angular: https://angular.io/guide/quickstart
Apache HTTP Server Version 2.4 Documentation: http://httpd.apache.org/docs/2.4/
Node.js: https://nodejs.org/en/docs/