import { HttpModule } from '@angular/http';

@NgModule({

declarations: [

AppComponent,

CarsComponent

],

imports: [

BrowserModule,

HttpModule

],

providers: [],

bootstrap: [AppComponent]

})

export class AppModule { }

# [How to bundle an Angular app for production](https://stackoverflow.com/questions/37631098/how-to-bundle-an-angular-app-for-production)

## OneTime Setup

* npm install -g @angular/cli
* ng new projectFolder creates a new application

## Bundling Step

* ng build --prod (run in command line when directory is projectFolder)

flag *prod* bundle for production (see the [*Angular documentation*](https://github.com/angular/angular-cli/wiki/build) for the list of option included with the production flag).

* Compress using [Brotli compression](https://en.wikipedia.org/wiki/Brotli) the resources using the following command

for i in dist/\*; do brotli $i; done

bundles are generated by default to ***projectFolder/dist(/$projectFolder for 6)***

## Output

Sizes with Angular *7.1.2* with CLI *7.1.2*and option CSS without Angular routing

* dist/main.[hash].bundle.js Your application bundled [ size: 174 KB for new Angular CLI application empty, **43 KB** compressed].
* dist/polyfill.[hash].bundle.js the polyfill dependencies (@angular, RxJS...) bundled [ size: 37 KB for new Angular CLI application empty, **11 KB** compressed].
* dist/index.html entry point of your application.
* dist/inline.[hash].bundle.js webpack loader
* dist/style.[hash].bundle.css the style definitions
* dist/assets resources copied from the Angular CLI assets configuration

## Deployment

You can get a preview of your application using the ng serve --prod command that starts a local HTTP server such that the application with production files is accessible using [http://localhost:4200](http://localhost:4200/).

For a production usage, you have to deploy all the files from the dist folder in the HTTP server of your choice.

# [@Directive v/s @Component in Angular](https://stackoverflow.com/questions/32680244/directive-v-s-component-in-angular)

Directives add behaviour to an existing DOM element or an existing component instance. One example use case for a directive would be to log a click on an element.

import {Directive} from '@angular/core';

@Directive({

selector: "[logOnClick]",

hostListeners: {

'click': 'onClick()',

},

})

class LogOnClick {

constructor() {}

onClick() { console.log('Element clicked!'); }

}

Which would be used like so:

<button logOnClick>I log when clicked!</button>

### Components

A component, rather than adding/modifying behaviour, actually creates its own view (hierarchy of DOM elements) with attached behaviour. An example use case for this might be a contact card component:

import {Component, View} from '@angular/core';

@Component({

selector: 'contact-card',

template: `

<div>

<h1>{{name}}</h1>

<p>{{city}}</p>

</div>

`

})

class ContactCard {

@Input() name: string

@Input() city: string

constructor() {}

}

Which would be used like so:

<contact-card [name]="'foo'" [city]="'bar'"></contact-card>

ContactCard is a reusable UI component that we could use anywhere in our application, even within other components. These basically make up the UI building blocks of our applications.

### In summary

Write a component when you want to create a reusable set of DOM elements of UI with custom behaviour. Write a directive when you want to write reusable behaviour to supplement existing DOM elements.