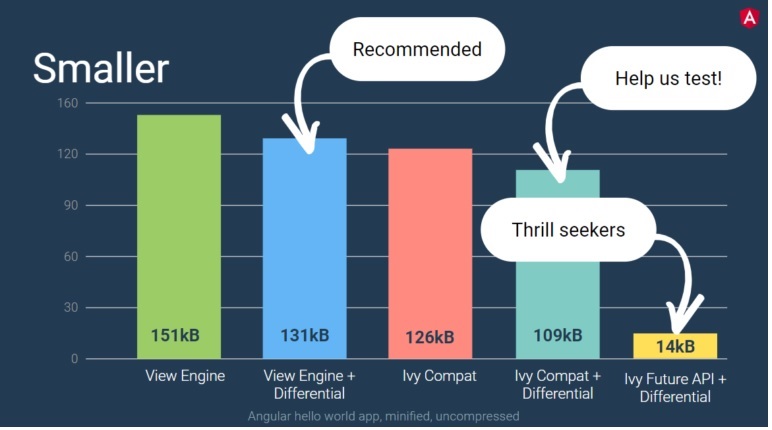
Angular 8.0 – What is new in Angular8

As we know that Google has just released its new version of Angular, i.e., 8.0 just a few days. Angular developers and community eagerly waited for the release of this version from a long time because, as announced by Google at the time of Angular 7 release, Angular 8 will be released with the features of IVY engine. So, the expectation is very high. Angular 8 is the first major release from Google in the year 2019 which is mainly focused on the toolchain and also making Angular easier for users to create different types of application along with performance improvements. Except this, this major version release also contains some new features and upgradation with respect to the previous versions. At the final stage, this release confirms that the new Angular version is much lighter, faster and easier. Also, Angular 8 supports the TypeScript version 3.4. So, with the help of the new TypeScript version, it is very much easy to code with faster subsequent build with the incremental flag, type checking for globalThis, and generic type arguments.

Let’s understand major changes made in Angular 8.

1. TypeScript 3.4.x Support: Angular 8 supports the TypeScript 3.4 or above version. So, if we want to use Angular 8 for our application, then we need to first upgrade the TypeScript to 3.4 or above.
2. Ivy Rendering Engine: The most important and expected feature of Angular 8 is IVY render engine. Ivy is the new Angular Compiler as well as a tool that acts as a new rendering pipeline. The benefit of Ivy is that it generates considerably small bundles, also can perform incremental compilation easily. So, Ivy is the basis of future innovations in the Angular world. Since in the last few releases many sections in the Angular have been changed, after switching to Ivy, the existing application will just work as same as earlier. But this will reduce the size of the bundles. In Angular 8, Google introduced a preview version of Ivy. The main objective of this version is to receive early feedback from the Angular Developer community related to Ivy. So, it is recommended to not use Ivy for the production environment right now as per the below image.
   1. Bundle sizes of a hello world application with and without Ivy (source: ngconf 2019 Keynote by Brad Green and Igor Minar)



* 1. At ngconf 2019, Brad Green, the Technical Director of the Angular Team at Google said that Ivy will allow a noticeable improvement related to the bundle size in compatibility mode in combination with differential loading. We will get the below benefits if we use Ivy in any application.
     1. It will provide the faster compilation (probably released in Angular 9)
     2. Much more improved type checking in templates so that we can catch more error at the build time and prevent the user to face those error at runtime. (probably released in Angular 9).
     3. Smaller bundle size compared to the current compiled bundled size.
     4. Also, provide backward compatibility since as rumors Google already used more than 600 application which is developed in Angular.
     5. Also, the code generated by the Angular compiler is now much easier human to read and understand.
     6. The last but most favorite feature of me is that we can debug the templates. I am quite sure these features will be liked by many developers like me.
  2. So, if we want to include the Ivy for a new application, we can create new projects with enable-ivy options

ng new ivy-project --enable-ivy

This command intimate Angular CLI store the following configuration in the tsconfig.json file.

"angularCompilerOptions": {

"enableIvy": true

}

* 1. The above configuration can manually add in any existing angular application after the upgrade that application into Angular 8.
  2. One recommendation that if we want to use Ivy for our application, then run the application in debug mode with AOT compilation mode.

ng serve --aot

Bazel Support

1. In Angular 8, Google introduced another build tool called Bazel. Actually, it is not a new tool. Google used this tool internally since a long time and now, they released this tool as open source. But one thing needs to clear that Bazel is not totally ready in Angular 8. It is introduced as an opt-in option with Angular 8 and is expected to be included in the Angular CLI in version 9. We can achieve the below benefits by using this tool,
   1. It will provide a faster build time. It will normally take time for the first build but taking less time from the concurrent builds.
   2. Using this tool, we can able to build the application as an incremental build and deploy only what has been changed rather than the entire app.
   3. We can eject the Bazel file which is normally hidden.
   4. We can add Bazel Support using the below Angular CLI Command,
      1. ng add @angular/bazel
   5. Also, we can create a new app with Bazel with the help of below command,
      1. npm install -g @angular/bazel
      2. ng new my-app --colection=@angular/bazel
2. Differential Loading for Performance Optimization: It is one of the cool new features in the Angular CLI 8. Because with the help of these features, we will specify which browser we will target and the CLI will build the application with related necessary JS bundles with the necessary polyfills. The default target browser in the tsconfig.json file is now es2015. It means when CLI build the application, it will build for the modern browser which supports ES6 features. But, we need to run the application in an older browser like IE9, then we need to specify it in the browserlist file. This file actually exists in the root folder of the CLI project and previously, it is used for the CSS part only. Now, it can be used for JS generation.
   1. # For IE 9-11 support, please remove 'not' from the last line of the file and adjust as needed

> 0.5%

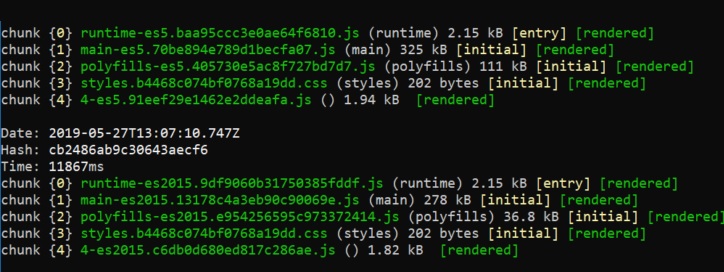
last 2 versions

Firefox ESR

not dead

not IE 9-11

So, as per the instruction above, if we remove the not keyword from the last line and then run the build process. So, the CLI receives an instruction from this file and bundling process will generate the files for both the versions.

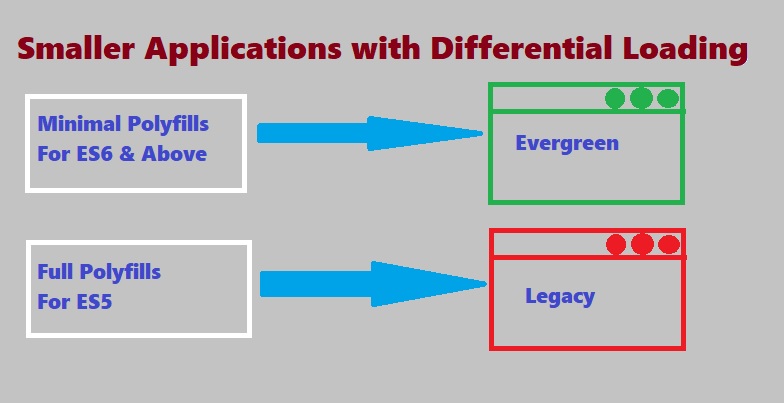


Now, only one disadvantage of this process is that it will take double build time. The different browsers can now decide which bundle need to load. For this purpose, a script reference has been added in the index.html file.

<script src="main-es2015.js" type="module"></script> -- For ES6 support browser or latest browser

<script src="main-es5.js" nomodule></script> -- For es5 support browser or old browser

In the above image, we can see that the bundle size significantly reduced in case of ES2015 version from 9% to 20%. So, ultimately small bundle size means performance optimization. Therefore, angular will build the additional files with polyfills and they will be included in the application with nomodules attribute.



1. Changes in Lazy Loading in Route: In Angular from the beginning, the router mechanism always supports the concept of lazy loading. Till Angular 7, it was done by the string value of the loading module as below,

{

path: 'lazy',

loadChildren: () => './admin/admin.module#AdminModule'

}

The value up to the #(hash) sign represents the path of the module file in which the target component exists and the value after the hash sign represent the class name of that module. This style will still work in Angular 8. But the way of writing lazy module has been changed. The string value of the loadchildren has been deprecated due to the support of ECMAScript and Ivy will only support this. So, now the loadchildren declaration will be like this,

{

path: 'lazy',

loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule)

}

1. Web Worker Support: As we all know that JavaScript is always executes in a single-threaded manner. So, it is important to perform any huge data call or any consecutive Rest API call in an asynchronous manner. But, in real application based scenario, this concept will not help us. That’s why today all the web browser support the web worker process. Basically, the web worker process is the scripts executed by the browser in a separate thread. Communicate with that thread in the browser tab will be done by sending messages. So, in general, web worker is not related to Angular from any point-of-view. But, the main point is that these scripts need to be considered in the build process of the application. So, that after deployment of application the same process can be performed continuously. So, the main objective is to provide one single bundle for every web worker. Now, in Angular 8 this task can be performed by the Angular CLI.

Also, we can configure our Angular CLI project if we add the web worker for the first time. In this process, CLI will exclude the worker.ts files from the tsconfig.json files and also, add a new TypeScript configuration file named tsconfig.worker.json which handles the worker.ts file. Also, this information also added to the angular.json file as

"webWorkerTsConfig": "tsconfig.worker.json"

1. Use Analytics Data: Now in Angular 8, Angular CLI collect usage analytics data so that Angular team can prioritize the features and improvements. So, when we update the CLI projects, it will opt-in with ng analytics on options. If we allow this globally, then it will collect some data like command used, the flag used, Operating System, Node Version, CPU Count, RAM Size, execution time and error with crash data if any to the Angular team for the improvement purpose in the future releases.
2. Bye Bye @angular/http: From Angular 8 version, angular stop the support for @angular/http. They had already depreciated the use of @angular/http in Angular 4 and provide an efficient and secure HTTP call by using @angular/common/http. But, still Angular 7, they allow us to use @angular/http in the application. But, from Angular 8 onwards, this support is not available further. So, we need to make adjustment in our code to use @angular/common/http in place of @angular/http.
3. Changes in ViewChild and ContentChild: In Angular 8, there are some breaking changes occurred in the use of ViewChild and ContentChild. It is an unexpected behavior from the Angular team in respect of the previous releases. Now, in Angular 8 it is mandatory to provide a Boolean static flag to define the instance of a ViewChild and ContentChild. Since already we experience an issue that sometimes these instances are available in ngOnInit, but also sometimes we didn’t find these instances until ngAfterContentInit or ngAfterViewInit. It occurred mainly due to elements that will be loaded into the DOM at a later stage according to some data binding (like as per \*ngIf or \*ngFor), then that elements had to be inserted into DOM in ngAfterViewInit or ngAfterContentInit. So, it is quite confusing and that’s why now we need to specify the component when the component resolution need to take place:-

@ViewChild('info', { static: false }) \_cityViewChild: CityComponent;

If the static value is true, angular will try to find the element at the time of component initialization i.e. ngOnInit. It can be done when the element not used in any structural directives. When the static value is false, angular will find the element after initializing the view.

1. Support SVG Template: Now, Angular 8 supports the template features with a file extension SVG. So, it means we can use the SVG extension file as a template in place of HTML file without any extra configuration setting. But a question is why we will use a .svg file as a template instead of using the image in an HTML file? The reason is when we use SVG as a template, then we can use that as a directive and as a result, we can bind it just like HTML templates. With this approach, we can dynamically generate interactive graphics in our Angular Applications.

@Component({

selector: "app-icon",

templateUrl: "./icon.component.svg",

styleUrls: ["./icon.component.css"]

})

export class DashboardComponent {...}

1. PNPM Support: In Angular 8, Angular CLI also support new package manager PNPM including NPM and Yarn. Also, in the command ng add now provide a new flag called - - registry for adding packages from any private NPM registry. This command is already available in the Angular CLI version for ng update command
2. Support Codelyzer 5.0: In Angular 8, the default TSLint configuration supports the Codelyzer 5.0 version. In this version of Codelyzer, some rule has been renamed. But when we upgrade the Angular CLI, it will also update the TSLint configuration file. The CLI update schematic will remove the es6 imports command from the polyfills.ts file. Because now it is automatically added if it required by the CLI.

1. Support for New Builders/Architect API: The new version of the Angular CLI allows us to use the new version of the Builders which is also known as Architect API. Angular used Builders API for performing the operations like server, build, test, lint and e2e. We can use builders in the angular.json file.

"projects": {

"app-name": {

"architect": {

"build": {

"builder": "@angular-devkit/build-angular:browser",

},

"serve": {

"builder": "@angular-devkit/build-angular:dev-server",

},

"test": {

"builder": "@angular-devkit/build-angular:karma",

},

"lint": {

"builder": "@angular-devkit/build-angular:tslint",

},

"e2e": {

"builder": "@angular-devkit/build-angular:protractor",

}

}

}

}

1. How to Upgrade in Angular 8: To upgrade any existing Angular 7 application to Angular 8, we need to perform the below steps,
   1. Angular 8 user TypeScript 3.4, so install TypeScript 3.4
   2. You need to use Node 12 or Later
   3. Now execute the upgrade command of Angular CLI -> ng update @angular/cli @angular/core
   4. Or, Alternatively, you can check your upgrade process from the below URL links - https://update.angular.io/