**Updated features in Angular 13**

1. **View Engine is no longer available.**

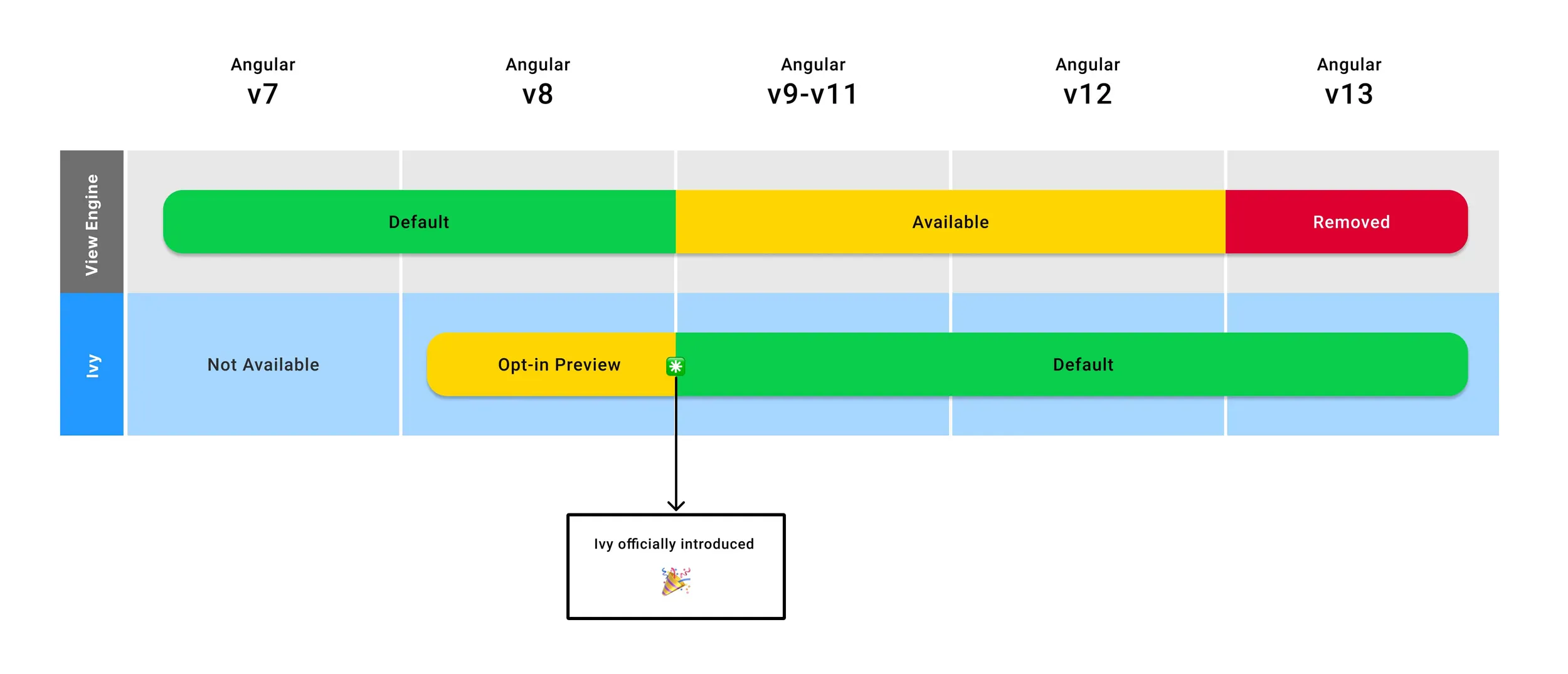
* The legacy View Engine is no longer supported by latest version, Angular 13.

Relying on View Engine has its maintenance costs, and also increases the complexity of Angular 13 codebases. To avoid that hassle, Ivy is now the only view engine supported by Angular.

Some existing libraries will be automatically migrated to **partial** compilation mode due to this change, and some metadata previously required for the legacy View Engine will also be removed. Furthermore, all internal tools are converted to Ivy ahead of time to ensure a smooth transition.

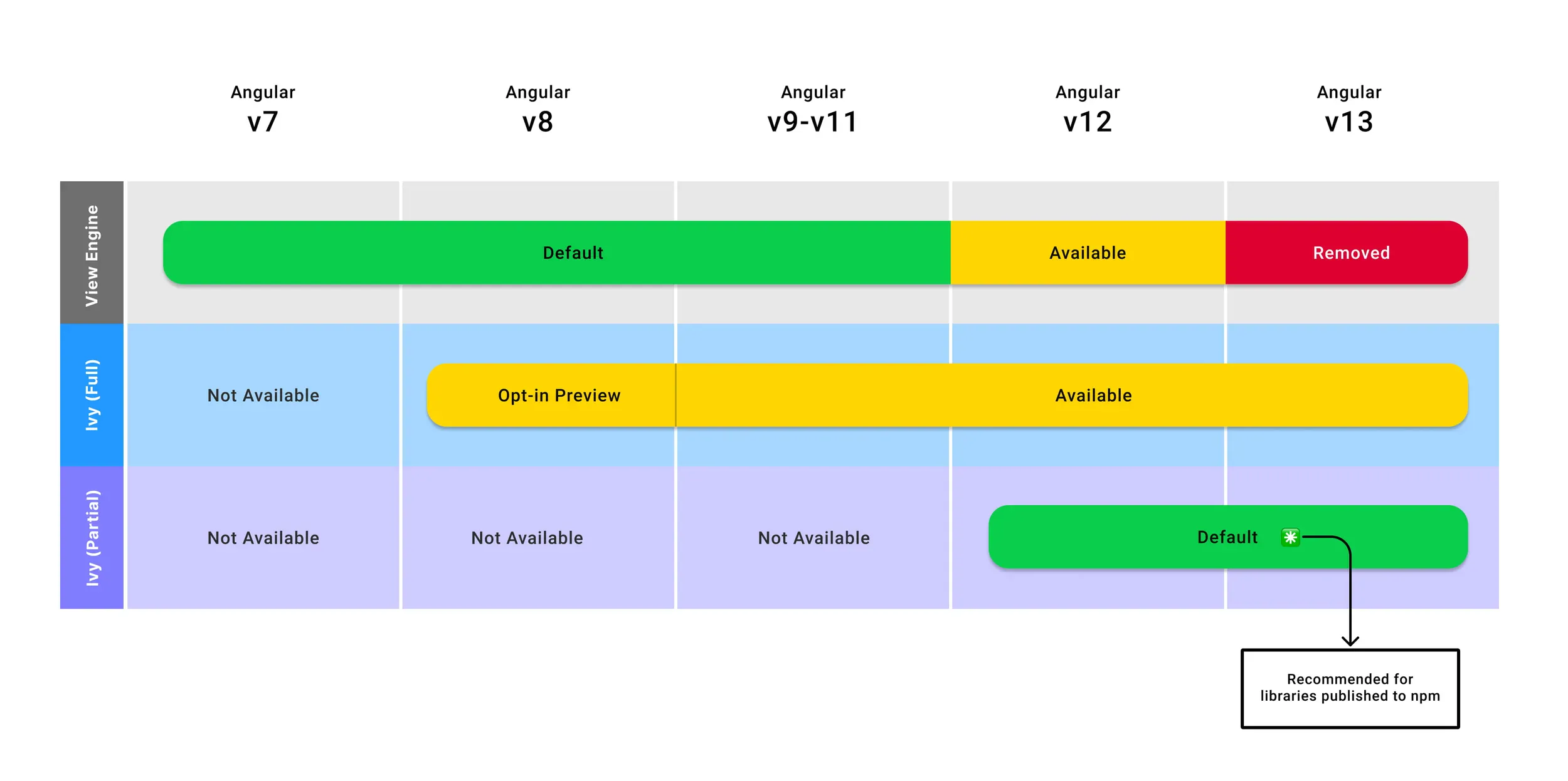
Ivy can now compile individual components independently of one another, which significantly improves performance. By removing View Engine, Angular can reduce its reliance on ngcc too.

**For Application developers:-**



**For Library Developers:-**

For library developers, especially the ones who publish to npm, the goal was always to support both engines during the transition. This was achieved by publishing in the View Engine format. View Engine apps could therefore still use it and Ivy apps had to run ngcc to make it compatible.

****

**With Angular 13, it is not possible to publish in the View Engine format anymore. Library developers have now two options to publish:**

1. Partial Ivy

This is the default when Angular CLI creates a library project. It creates an intermediate format. Since it is only partially compiled it is compatible with any Angular versions higher than 12. It gets fully compiled when the app is built.

1. Full Ivy

Fully compiled Ivy code. Not recommended when publishing the library. Use it in your application workspace where all apps and libs share the same Angular version

1. **Changes to the Angular Package Format (APF)**

* The Angular Package Format (APF) defines the format and structure of Angular Framework packages and View Engine metadata. It’s an excellent strategy for packaging every third-party library in the web development environment.

We can see some significant changes in the new version of the APF. Older output formats, including some View Engine-specific metadata, are removed with Angular 13.

The updated version of APF will no longer necessitate the use of ngcc. As a result of these library changes, developers can expect faster execution.

## Component API updates

Ivy also enables quality of life improvements to the way developers can dynamically create components. The API has now been simplified. Before the changes in Angular v13, dynamically creating components required a lot of boilerplate code.

The new API removes the need for [**ComponentFactoryResolver**](https://v13.angular.io/api/core/ComponentFactoryResolver) being injected into the constructor. Ivy creates the opportunity to instantiate the component with **[ViewContainerRef.createComponent](https://v13.angular.io/api/core/ViewContainerRef" \l "createComponent" \t "_blank)** without creating an associated factory.

**Here’s an example of creating components with previous versions of Angular:**

.

|  |
| --- |
| @Directive({ … }) |
|  | export class MyDirective { |
|  | constructor(private viewContainerRef: ViewContainerRef, |
|  | private componentFactoryResolver: |
|  | ComponentFactoryResolver) {} |
|  | createMyComponent() { |
|  | const componentFactory = this.componentFactoryResolver. |
|  | resolveComponentFactory(MyComponent); |
|  |  |
|  | this.viewContainerRef.createComponent(componentFactory); |
|  | } |
|  | } |

**With the new API, this code can become:**

|  |
| --- |
| @Directive({ … }) |
|  | export class MyDirective { |
|  | constructor(private viewContainerRef: ViewContainerRef) {} |
|  | createMyComponent() { |
|  | this.viewContainerRef.createComponent(MyComponent); |
|  | } |
|  | } |

# **End of IE11 support**

Angular will no longer support Internet Explorer 11 as of version 13. Dropping IE 11 is a positive factor because it results in smaller bundle size and faster app loading. In addition, Angular can now use modern browser features like CSS variables and web animations via native web APIs due to these enhancements.

Removing IE11 support allows Angular to leverage modern browser features such as CSS variables and web animations via native web APIs. What’s more is that apps will be smaller and load faster because we can remove IE specific polyfills and code paths. It also removes the need for [differential loading](https://v12.angular.io/guide/deployment#differential-loading). Developers will benefit from improved APIs and build infrastructure while application users will benefit from faster loading and an improved user experience.

During project migration, running **ng update** will automatically remove these IE-specific polyfills and reduce the bundle size.

## 4. TypeScript 4.4 support.

## TypeScript 4.4 support is now available in Angular 13 version. As a result, versions prior to TypeScript 4.4.2 are no longer supported in the core.

## Developers who still need to support IE11 users for existing projects can continue to use Angular v12 and it will be [supported until November 2022](https://angular.io/guide/releases#support-policy-and-schedule).

### **The significant highlights of TypeScript 4.4 are:**

* Improved detection of type guards.
* Default catch variables.
* Faster incremental builds.
* The control flow of conditions can be analyzed.
* Symbol and template string pattern index signatures

1. **Improvements to the Angular CLI**

On to the updates to **[Angular’s tooling](https://angular.io/cli" \t "_blank)**. Angular now supports the use of persistent build cache by default for new v13 projects. The valuable feedback from [**[RFC]** **Persistent build cache by default**](https://github.com/angular/angular-cli/issues/21545) led to this tooling update that results in up to 68% improvement in build speed and more ergonomic options. In order for existing projects that have been upgrading to v13 to enable this features developers can add this configuration to angular.json:

|  |
| --- |
| { |
|  | "$schema": "...", |
|  | "cli": { |
|  | "cache": { |
|  | "enabled": true, |
|  | "path": ".cache", |
|  | "environment": "all" |
|  | } |
|  | } |
|  | ... |
|  | }  ESBuild also sees some performance improvements in this release! We introduced [esbuild](https://esbuild.github.io/" \t "_blank), which now works with [terser](https://terser.org/) to optimize global scripts. In addition, esbuild supports CSS sourcemaps and can optimize global CSS, as well as optimizing all style sheets.  **6. Framework changes and dependency updates**  Angular v13 also features some helpful updates and important changes. First up, RxJS 7.4 is now the default for apps created with ng new. Existing apps using RxJS v6.x will have to manually update using the npm install rxjs@7.4 command.  **7. Improvements to Angular tests.**  We’ve made some important improvements to [TestBed](https://v13.angular.io/api/core/testing/TestBed" \t "_blank) that now does a better job of tearing down test modules and environments after each test. The DOM is now cleaned after every test and developers can expect faster, less memory-intensive, less interdependent, and more optimized tests.  This feature has been opt-in since 12.1.0 and now it’ll be the default while remaining customizable. Here’s how - it can be configured for the entire test suite via the [TestBed.initTestEnvironment](https://v13.angular.io/api/core/testing/TestBed#initTestEnvironment) method:   |  | | --- | | beforeEach(() => { | |  | TestBed.resetTestEnvironment(); | |  | TestBed.initTestEnvironment( | |  | BrowserDynamicTestingModule, | |  | platformBrowserDynamicTesting(), | |  | { | |  | teardown: { destroyAfterEach: true } | |  | } | |  | ); | |  | }); |   Or it can be configured per module by updating the [TestBed.configureTestingModule](https://v13.angular.io/api/core/testing/TestBed" \l "configureTestingModule" \t "_blank) method:   |  | | --- | | beforeEach(() => { | |  | TestBed.resetTestEnvironment(); | |  | ... | |  | TestBed.configureTestingModule({ | |  | declarations: [TestComp], | |  | teardown: { destroyAfterEach: true } | |  | }); | |  | }); |  8. Node.js versions prior to 12.20 are no longer supported.Node.js versions prior to 12.20.0 are no longer available in Angular 13. This is because the Angular packages now use the Node.js package export feature with subpath patterns. |

## 9. Version 7.4 of RxJS

## The Angular 13 update adds RxJS, a reactive extension for JavaScript, and includes all versions of RxJS up to and including version 7.

## For apps created with ****ng new****, RxJS 7.4 has become the default.

## Existing RxJS v6.x apps will need to be manually updated with the **npm install rxjs@7.4** command. You can always rely on RxJS 7 for new project creation. As for migrations, existing projects should keep on RxJS 6.

## 10. Router changes

## Routing help us to handle navigation from one view to the another. It enables easy navigation by interpreting a browser URL as an instruction to change the view.

## With the latest update, the router no longer replaces the browser URL when new navigation cancels the current navigation.

## There were several compatibility issues with earlier versions of Angular, mainly with query parameters. For example, after a question mark in query parameters, the default URL serializer tends to drop everything. The latest update, on the other hand, improves query parameter compatibility with question marks.

## The null and undefined inputs of ****routerLink**** directives were previously equivalent to an empty string, and there was no mechanism to block the navigation of the link. Setting the router link directive value to null or undefined will now completely disable the navigation.

## 11. Creating dynamic documents.

## One Ivy-enabled API update in Angular 13 is a more streamlined method for dynamically constructing a component. ****ViewContainerRef.createComponent**** no longer requires an instantiated factory to construct a component (you no longer need to use **ComponentFactoryResolver**).

## Due to the improved **ViewContainerRef.createComponent** API, it is now possible to create dynamic components with less boilerplate code. Here’s an example of creating dynamic components using previous versions of Angular.

@Directive({ … })

export class Test {

constructor(private viewContainerRef: ViewContainerRef,

private componentFactoryResolver:

ComponentFactoryResolver) {}

createMyComponent() {

const componentFactory = this.componentFactoryResolver.

resolveComponentFactory(MyComponent);

this.viewContainerRef.createComponent(componentFactory);

}

}

In Angular 13, this code can become the following using the new API.

@Directive({ … })

export class Test {

constructor(private viewContainerRef: ViewContainerRef) {}

createMyComponent() {

this.viewContainerRef.createComponent(MyComponent);

}

}

# **12. Other notable updates!**

With the release of [Angular v11](https://blog.angular.io/version-11-of-angular-now-available-74721b7952f7) in 2020, we introduced support for inlining Google Fonts. Now, we’re extending support to [Adobe Fonts](https://fonts.adobe.com/). Remember, inlining fonts can improve your app performance by speeding up the [First Contentful Paint](https://web.dev/first-contentful-paint/) (FCP). This change is now enabled for everyone by default! All you need to do is ng update.