Angular application can be compiled using two following commands:-

1. ng-serve
2. ng-build

The ng-serve and [ng-build](https://angular.io/cli/build) commands provide the development build by default and bundle all the files to a single file.

The similarities between these two commands are:-

* Both the commands compile the application and produce the development build by default. In this stage, compiled files don't have any optimization done by it.
* Both the commands do the bundling of all the files present to the single file. Bundling is the process of combining all the files into a single file. It produces 5 types of bundled [javascript](https://www.javascript.com/" \t "_blank) files and the source map files by default. These files are embedded in an index.html file loaded by the browser.

A fast cluster computing platform developed for performing more computations and stream processing. Click to explore about, [Apache Spark Optimization Techniques](https://www.xenonstack.com/blog/apache-spark-optimisation)

The files which get generated are:-

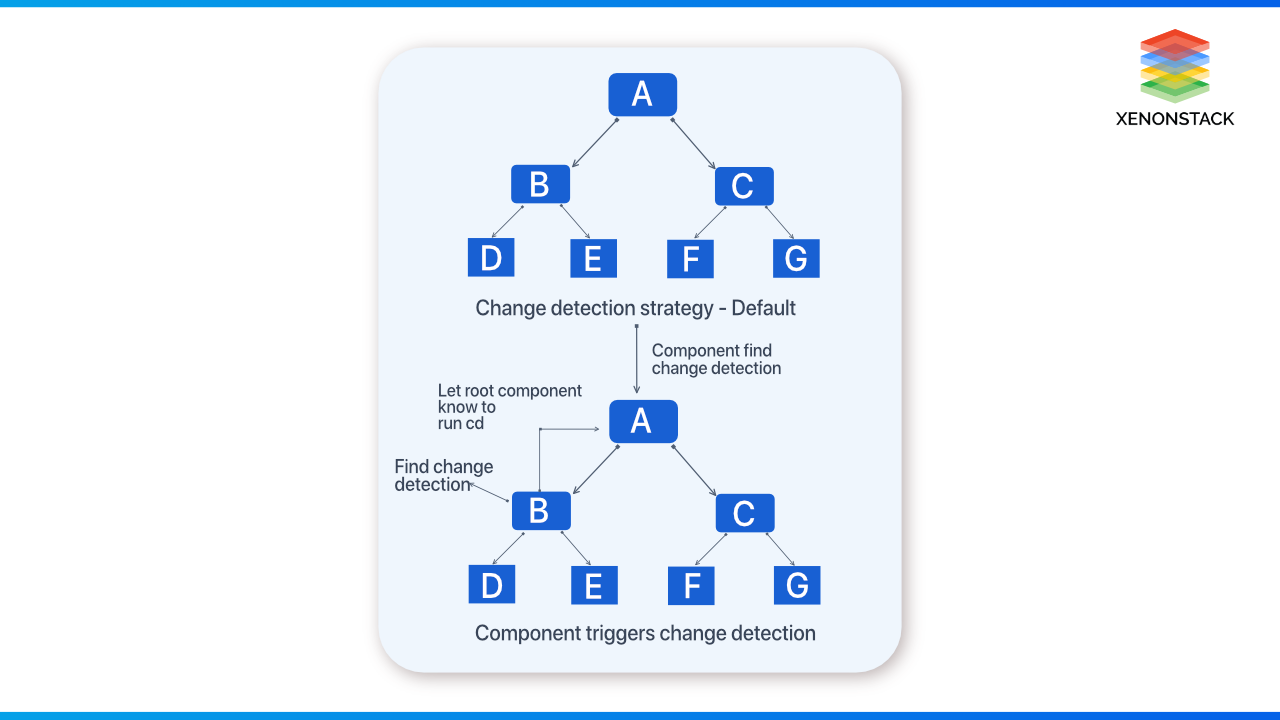
* **Inline.bundle.js**:- It contains the script that is responsible for the webpack to run.
* **polyfills.bundle.js**:- It contains the script responsible for making the application compatible with all the browsers.
* **main.bundle.js**:- It contains the code present in the application.
* **styles.bundle.js**:- It contains the styles used by the application.
* **Vendor.bundle.js**:- It contains the angular and includes 3rd party libraries.

The differences between the two commands are:-

* **ng-serve**:- It compiles and runs the application from memory and is used for the development process. It doesn't write any files to the build folder. So, It can't be deployed to another server.
* **Ng-build**:- It compiles and produces the build files to an external folder. As a result, this created folder(build) is deployed in any external server. The name of the build folder is decided by the value of the [outputPath](https://docs.aws.amazon.com/step-functions/latest/dg/input-output-outputpath.html" \t "_blank) property present in the build section of the angular.json file

It provides two modes of compilation:-

* AOT(Ahead Of Time) Compilation
* JIT(Just In Time) Compilation

Until Angular 8, the default compilation mode was JIT, but from angular 9, the default compilation is AOT. When we do ng serve, it depends on the value of aot passed in the angular.json file.

**JIT**

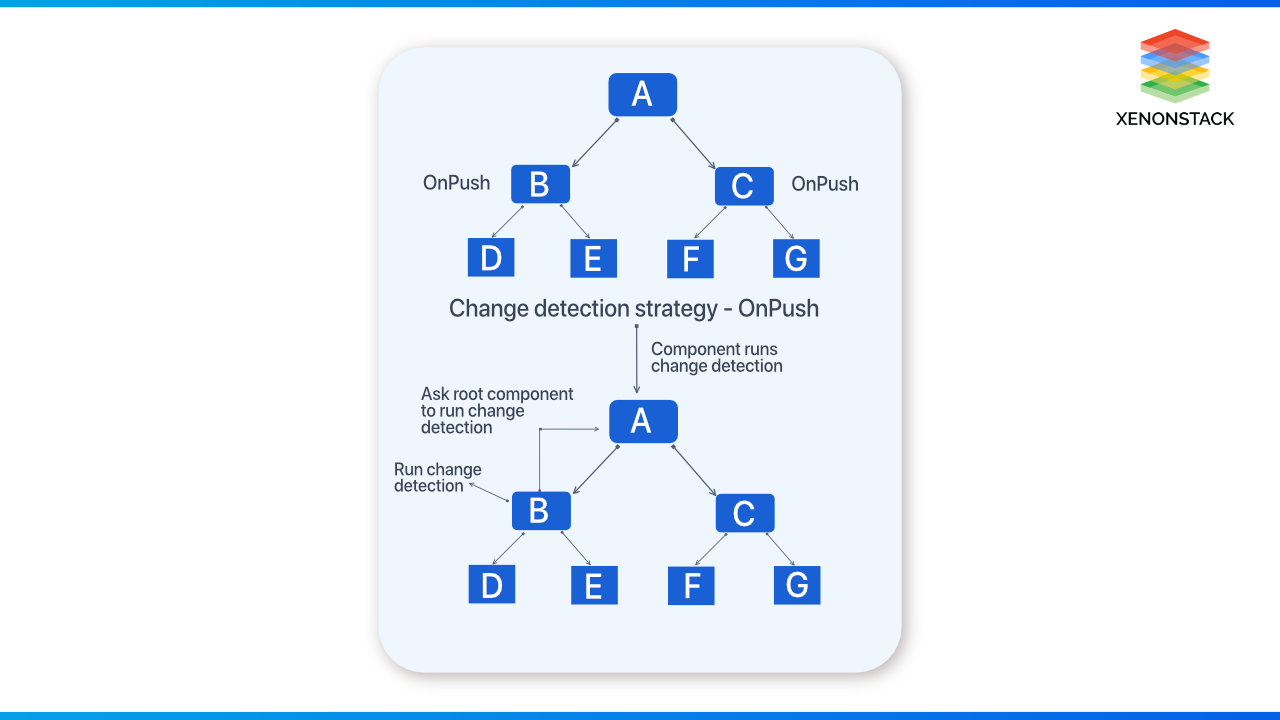
The JIT mode compiles the application in the application's runtime. The browser downloads the compiler along with the project files. The compiler is about 45% of the vendor.bundle.js file loaded in the browser.

The disadvantages of the JIT compiler is that:-

* It increases the application size due to the compiler in the browser, which affects the overall application's performance.
* The user has to wait for the compiler to load first and then the application. Thus, it increases the load time.
* Also, the template binding errors are not shown in the build time.

**AOT Compilation**

In [AOT compilation](https://angular.io/guide/aot-compiler), the compilation is done during the build process, and the compiled files are processed are bundled in the vendor. bundle.js file, which gets downloaded by the browser. The compilation is during the build time, so the size decreases by 50%.



The major advantages of the AOT compilation are:-

* The application's rendering becomes faster as the browser downloads only the pre-compiled version of the application.
* The application has a small download size because it doesn't include the compiler with itself, as the compiler takes half of the actual size of the application.
* Also, the template binding errors were detected during the build time of the application. Hence, it helps to fail fast and makes the development process easy.

The performance of any application plays a vital role in the growth of the business. As it is a high-performing [front-end framework](https://github.com/collections/front-end-javascript-frameworks), we face challenges optimizing the performance. The major performance issues are the decline in traffic, decrement in engagement, high bounce rate, and crashing of applications in high usage.

The performance of any application can only be optimized by knowing the exact reason or identifying the issue for degradation of the performance. Some of the problems faced by the application are:-

* Unnecessary server usage
* Slow page response
* Unexpected errors due to a real-time data stream
* Periodic slowdown

These problems can be solved using proper techniques. Also, we should ensure that we have followed the best coding practices and clean code architecture. We (In Xenonstack) have improved the application's performance by following this.

Few other solutions to optimize the performance issues would be:-

* Removing the unnecessary change detection in the app slows down the application.
* Adding the onPush at required places.
* Reducing the computation on the template side.

The framework does its job well to provide the best performance. But, if we want to develop a large-scale application, we need to deal with many other factors. A few popular methods or techniques help us optimize the application better.

So, Let's start by going through the few methods in detail. These are a few essential hacks that can help us significantly alleviate the performance.

### Using AoT Compilation

As we know that it provides two types of compilation:-

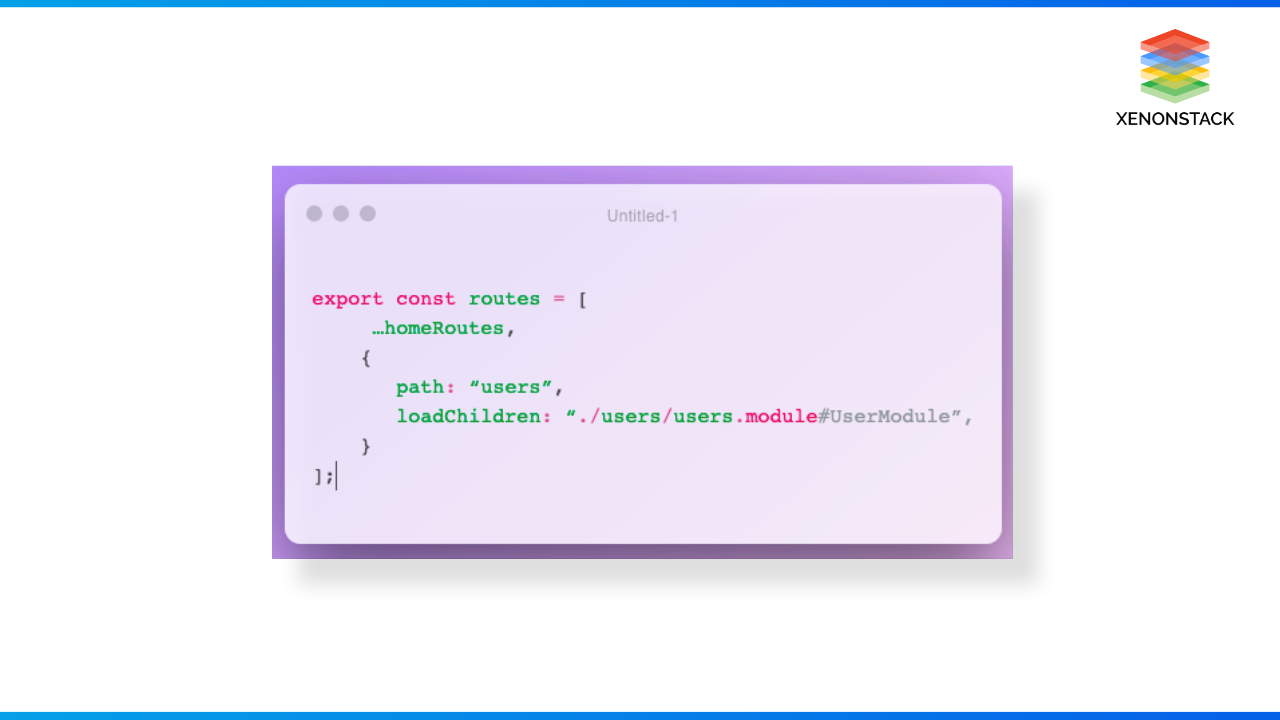
* JIT(Just-in-time)
* AoT(Ahead-of-time)

We also looked at the compilation's working in the above section. Afterward 8 version provides the AoT compilation, which increases the performance. Because the JIT compiles the application in the runtime. Also, the [JIT compilation](https://www.ibm.com/docs/en/sdk-java-technology/8?topic=reference-jit-compiler) bundles the compiler with itself, which increases the size of the bundler. Also, it increases the rendering time of the component.

But with the AoT compilation compiles the application in the build time, produces only the compiled templates, and doesn't include the compiler. So, the bundle size decreases, and rendering time increases significantly. So, We should always use AoT compilation for our applications.

### Using OnPush Change Detection Strategy

Before deep-diving into the concept of change detection strategy, it is essential to know what change detection is. Change detection is the process by which angular updates the DOM.

The change detection in the angular application looks something like the below diagram. It detects the changes in the application within the tree of components. It starts by checking the component's root, then children and their grandchildren. The only change in component B makes the change detection run all over the components, as shown below.

In the case of Reference type, i.e., objects, whenever some changes occur, Angular checks each property of the object to check if any property value has changed or not and updates the [DOM](https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction) accordingly. So, this affects the performance of the application.

This can be controlled and fixed using the OnPush detection strategy. It tells Angular not to check each component every time any change detection occurs.

The OnPush strategy makes our component smarter. It calls current component change detection only when the @input bindings change. It only runs change detection for descendant components. So, this helps to improve the performance of the application.

After applying OnPush change detection to Component B and C, once the change detection happens on component B, it triggers the tree's root to the bottom. And root component changes the input property of B. So, using the OnPush change detection method significantly changes the application's performance.

A non-functional type of testing that measures the performance of an application or software under a certain workload based on various factors. Click to explore about, [Performance Testing Tools](https://www.xenonstack.com/insights/what-is-performance-testing)

### Using Pure Pipes

In Angular, pipes transform the data to a different format. E.g., the:- 'date | short date converts the date to a shorter format like 'dd/MM/yyyy.' Pipes are divided into two categories:-

* Impure pipe
* Pure Pipe

The impure pipe is those pipes that produce different results for the same input over time. The pure pipes are the pipes that produce the same result for the same input. It has few built-in pipes which are pure. When it comes to binding evaluation, angular each time evaluates the expression and applies the pipe over it(if it exists). The Angular applies a nice optimization technique: the 'transform' method, which is only called if the reference of the value it transforms is changed or if one of the arguments changes. It caches the value for the specific binding and uses that when it gets the same value. It is similar to memorization. We can also implement the memorization of our own. It can be implemented like this

### Unsubscribe from Observables

Ignoring minor things in the development of the application can lead us to significant setbacks like memory leaks. Memory leaks occur when our application fails to eliminate the resources that are not being used anymore. Observables have the subscribe method, which we call with a callback function to get the values emitted. Subscribing to the observables creates an open stream, and it is not closed until they are closed using the unsubscribe method. Such as global declaration of variables creates unnecessary memory leaks and does not unsubscribe observables.

So, It is always a good practice to unsubscribe from the observables using the [onDestroy](https://angular.io/api/core/OnDestroy" \t "_blank) lifecycle hook, So when we can leave the component, all the used observables are unsubscribed.

### Lazy Loading

The enterprise application built using angular contains many feature modules. All these modules are not required to be loaded at once by the client. With large enterprise applications, the size of the application increases significantly with time, and it also increases the bundle size.

Once the bundle size increases, the performance goes down exponentially because every KB extra on the main bundle contribute to slower:-

* Download
* Parsing
* JS Execution

This can be solved using lazy loading. [Lazy Loading](https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading) is loading only the necessary modules at the initial load. This not only reduces the bundle size but also decreases the load time. Other modules are only loaded when the user navigates to the created routes. This increases the application load time by a great deal.

### Use trackBy option for For Loop

It uses \*ngFor directive to loop over the items and display them on the DOM(by adding or removing the elements). If it is not used with caution, it may damage its performance.

Suppose we have the functionality of adding and removing the employees asynchronously. For example, if we have a large object containing the list of all the employees with the name and email address, we need to iterate over the list and display the data on the UI. Then for each addition and removing of data, the angular change detection runs and sees if the object has some new value. If it finds, it destroys the previous DOM and recreates the DOM for each item again.

This has a huge impact on performance as rendering DOM is expensive. So, to fix this [trackBy](https://angular.io/api/core/TrackByFunction" \t "_blank) function is used. It keeps track of all the changes and only updates the changed values.

### Avoid computation in template files

The Template expressions are the most commonly used things in Angular. We often need a few calculations on the data we get from the backend. To achieve that, we use functions on the template.

The bound function runs when the CD(change detection) runs on the component. The function also needs to complete before the change detection and other codes move on.

If a function takes a long time to finish, it will result in a slow and laggy UI experience for users because it will stop other UI codes from running. It needs to be completed before the other code executes. So, It is must be needed that template expressions finish quickly. If highly computational, it should be moved to the great components file and the calculation beforehand.

### Usage of Web Workers

As we know that JS is a single-threaded language. This means that it can only run on the main thread. JS running in the browser context is called the DOM thread. When any new script loads in the browser, a DOM thread is created where the JS engine loads, parses, and executes the JS files on the page. If our application performs any such task that includes heavy computation at startup, like calculating and rendering the graphs, it increases the application's load time and we know that when any application takes >3 secs to load, the user switches to another application. Also, it leads to a horrible user experience.  
  
So, In such cases, we can use Web Workers. It creates a new thread called the worker thread that runs the script file parallel to the DOM thread. The worker thread runs in a different environment with no DOM APIs, so it has no reference to the DOM.  
So, using web workers to perform heavy computation and the DOM thread for light tasks helps us achieve greater efficiency and decreased load time.