>To generate a development build we can use either

ng build OR ng build --dev

>To generate a production build we use

ng build --prod

>Here are some of the differences between a development build and a production build in angular.

>Source Maps : Development build generate Source Maps where as production build does not.

>What are Source Maps: To improve the performance, the application's JavaScript and CSS files are combined and compressed. It is extremely difficult to debug those compressed files. A source map holds information about the original files and can be used to map the code within a compressed file back to it’s original position in a source file. So with the help of these source maps we can easily debug our applications even after the the files are compressed and combined. By default, a development build produce source maps where as a production build does not.

However, we can change this default behaviour by using --sourcemaps option along with the ng build command. It's alias is -sm.

The following command produces a development build without source maps as we have set -sm option to false

ng build --dev -sm false

On the other hand, if you want source maps along with your production build set -sm option to true as shown below.

ng build --prod -sm true

>Extracts CSS : With the development build global styles are extracted to .js files where as with the production build they are extracted to .css files. To change this default behaviour use --extract-css option or it's alias -ec with the ng build command.

The following command produces a development build with global styles extracted to .css file(s) instead of .js ones.

ng build --dev -ec true

>Minification & Uglification : A Prod Build is both minified and uglified, where as a Dev Build is not.

What is Minification

The process of removing excess whitespace, comments, and optional tokens like curly brackets and semicolons is called Minification.

What is Uglification

The process of transforming code to use short variable and function names is called uglification.

The minified and uglified version of the file is smaller in size than the full version, resulting in faster response times and lower bandwidth costs.

If you look at the bundles generated by the prod build, you will notice that they are minified and uglified. Notice, extra whitespaces, comments, and optional tokens like curly brackets and semicolons are removed. Also notice, the code is transformed by using short variable and function names. On the other hand, the bundles generated by the dev build, are not minified and uglified.

Tree Shaking : A Prod build is Tree Shaked, where as a Dev build is not.

What is Tree Shaking

Tree shaking is the process of removing any code that we are not actually using in our application from the final bundle. It's one of the most effective techniques to reduce the application size.

If you look at the bundles generated by the production build, they are significantly less in size compared with the bundles generated by the development build. This is beacause with the production build the code is tree shaked to remove dead code i.e the code that is not referenced by the application.

Ahead-of-Time (AOT) Compilation : With a production build we get AOT (Ahead-of-Time) compilation, i.e the Angular component templates are pre-compiled, where as with a development build they are not. We will discuss Ahead-of-Time compilation in detail in our next video.

The following table summarises the differences between a development build and a production build

Feature Development Build Production Build

Source Maps Yes No

Extracts CSS .js file .css file

Minifaction No Yes

Uglification No Yes

Tree Shaking No Yes

AOT No Yes

**AOT and JIT:**

In Angular we have 2 models of compilation

JIT - Just-in-Time Compilation : JIT compilation as the name implies, compiles the application Just-in-Time in the browser at runtime.

AOT - Ahead-of-Time Compilation : AOT compilation compiles the application at build time.

By default, with the development build we get JIT compilation. This is how it works. The application code along with the angular compiler is downloaded by the browser. At runtime, when a request is issued to the application, the JIT-compiler in the browser compiles the application code before it is executed. This means our user who made that first request has to wait for the application to compile first.

In our previous videos we have seen that, when we build our angular application, the following JavaScript bundles are generated.

Inline

Main

Polyfills

Styles

Vendor

The vendor bundle contains the compiler along with the angular framework. The compiler code is roughly half of the Angular framework.

There is a tool called source-map-explorer that we can use to inspect the JavaScript bundles. This tool analyzes the source map generated with the bundle and draws a map of all dependencies.

To be able to use this tool we have to install it first. To install this tool, execute the following command

npm install source-map-explorer --save-dev

Once we have the tool installed, if you have not done the development build yet, do the development build using the following command.

ng build

Once the build is complete, you will have the JavaScript bundles along with the source map files. Now execute the following command.

node\_modules\.bin\source-map-explorer dist\vendor.bundle.js

The above command runs the source-map-explorer against the vendor bundle and we see the graph of it. Notice the angular compiler is around 45% percent of the bundle size. As this is development build and not optimised, notice the total size of the bundle is 2.19 MB.

With AOT compilation the angular application is pre-compiled. So this means the browser loads executable code so it can render the application immediately, without waiting to compile the application first.

This also mean with AOT, as the application is already pre-compiled, there is also no need for the browser to download the Angular compiler. As we already know, the compiler code is roughly half of the Angular framework, so omitting it dramatically reduces the application size.

By default, the production build is Ahead-of-Time compiled. So there is no need to bundle up the angular compiler code in the vendor bundle. This brings down the vendor bundle size by almost 50%. In addition it is also minified, uglified and tree-shaked to remove any code that we are not referencing in our application. So the bundler size is further reduced.

Now, execute the following command to generate a production build. Notice I have also turned on sourcemap option. Without the sourcemap we will not be able to use the source-map-explorer tool.

ng build --prod --sourcemap true

Once the production build is complete, execute the following command. Vendor bundle name in your production build may be slightly different. Change it accordingly and execute the command.

node\_modules\.bin\source-map-explorer dist\vendor.7e385ef294695236ffd1.bundle.js

The AOT compiler also detects and reports template binding errors at build time itself. Let us understand this with an example.

Include the following function HomeComponent class in home.component.ts file

getText(): string {

return 'Hello Pragim';

}

In home.component.html include the following [div] element. Notice I have deliberately mis-spelled the getText() function name.

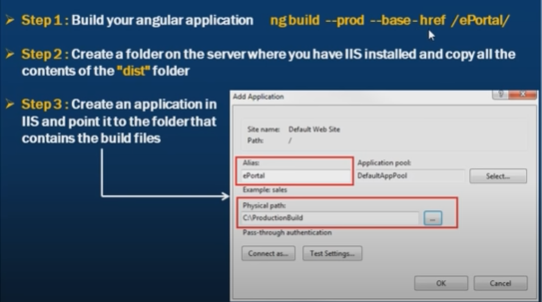
[div [innerText]='getTex()']

Save changes, and execute the following command. This command does a development build in-memory. At the moment we are not using AOT, so we will not know about the template binding error that is introduced above. Notice at build time we do not see any errors

**deploying angular application to IIS:**

we can deploy development build or production build into server.

But prod is the best because of less size of bundles(less size duto to performance techniques applied on it and also no source maps and aot)





ng build --base-href /ePortal/

If you want to deploy a production build do a production build using the following Angular CLI command.

ng build --prod --base-href /ePortal/

In our case let's deploy a production build. After the build is complete, you will notice a folder with name "dist" in your Angular project folder. This folder contains all the build files. These build files need to be copied to a folder on the server where we have IIS installed.

Step 2 : Create a folder on the server where you have IIS installed. You can name the folder anything you want. I am going to name the folder "ProductionBuild" and I am creating it in C:\ drive.

Step 3 : Now copy all the contents of the "dist" folder into "ProductionBuild" folder

Step 4 : Open IIS. There are several ways to do this. One way is to type "inetmgr" in the "Run" window and click "OK"

Step 5 : Create an application in IIS. Name it "ePortal". This name has to match the value we have specified for the --base-href option in Step 1.

a) Exapand the root IIS node

b) Expand Sites

c) Right click on "Default Web Site" and select "Add Application" from the context menu

d) In the "Alias" textbox, type "ePortal"

e) Set the "Physical Path" to folder that contains the build files. In our case it is "ProductionBuild" folder in C:\ drive

At this point, if you launch the browser and navigate to http://localhost/ePortal/home,

If you want to deploy a production build do a production build using the following Angular CLI command.

ng build --prod --base-href /ePortal/

In our case let's deploy a production build. After the build is complete, you will notice a folder with name "dist" in your Angular project folder. This folder contains all the build files. These build files need to be copied to a folder on the server where we have IIS installed.

Step 2 : Create a folder on the server where you have IIS installed. You can name the folder anything you want. I am going to name the folder "ProductionBuild" and I am creating it in C:\ drive.

Step 3 : Now copy all the contents of the "dist" folder into "ProductionBuild" folder

Step 4 : Open IIS. There are several ways to do this. One way is to type "inetmgr" in the "Run" window and click "OK"

Step 5 : Create an application in IIS. Name it "ePortal". This name has to match the value we have specified for the --base-href option in Step 1.

a) Exapand the root IIS node

b) Expand Sites

c) Right click on "Default Web Site" and select "Add Application" from the context menu

d) In the "Alias" textbox, type "ePortal"

e) Set the "Physical Path" to folder that contains the build files. In our case it is "ProductionBuild" folder in C:\ drive

At this point, if you launch the browser and navigate to http://localhost/ePortal/home, "dist" in your Angular project folder. This folder contains all the build files. These build files need to be copied to a folder on the server where we have IIS installed. Step 2 : Create a folder on the server where you have IIS installed. You can name the folder anything you want. I am going to name the folder "ProductionBuild" and I am creating it in C:\ drive. Step 3 : Now copy all the contents of the "dist" folder into "ProductionBuild" folder Step 4 : Open IIS. There are several ways to do this. One way is to type "inetmgr" in the "Run" window and click "OK" Step 5 : Create an application in IIS. Name it "ePortal". This name has to match the value we have specified for the --base-href option in Step 1. a) Exapand the root IIS node b) Expand Sites c) Right click on "Default Web Site" and select "Add Application" from the context menu d) In the "Alias" textbox, type "ePortal" e) Set the "Physical Path" to folder that contains the build files. In our case it is "ProductionBuild" folder in C:\ drive At this point, if you launch the browser and navigate to [http://localhost/ePortal/home](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbE9kS0UtUUdBYnk0cEM2Ym90X2JVUlRvaXFMd3xBQ3Jtc0trSU9kaEQ1ekRMMTRTWGVtSDRtdlQ0eGU3Y3JwM0JJcXhlY0JENXJDYV9Sam4xaWY4eV9GeloydVpEOERNYUtGUHZMLTFDSWkzUE1UUks5ZXFTM3c4MnpCVEhSZUlaakhQdGlBbzBLUl9KU3poMHRuWQ&q=http%3A%2F%2Flocalhost%2FePortal%2Fhome),