# Fundamentals

## Components & Templates: Dynamic component loader

Component templates are not always fixed. An application may need to load new components at runtime.

This cookbook shows you how to use [ComponentFactoryResolver](https://angular.io/api/core/ComponentFactoryResolver) to add components dynamically.

See the [live example](https://angular.io/generated/live-examples/dynamic-component-loader/stackblitz.html) / [download example](https://angular.io/generated/zips/dynamic-component-loader/dynamic-component-loader.zip) of the code in this cookbook.

## Dynamic component loading

The following example shows how to build a dynamic ad banner.

The hero agency is planning an ad campaign with several different ads cycling through the banner. New ad components are added frequently by several different teams. This makes it impractical to use a template with a static component structure.

Instead, you need a way to load a new component without a fixed reference to the component in the ad banner's template.

Angular comes with its own API for loading components dynamically.

## The anchor directive

Before you can add components you have to define an anchor point to tell Angular where to insert components.

The ad banner uses a helper directive called AdDirective to mark valid insertion points in the template.

src/app/ad.directive.ts

content\_copyimport { [Directive](https://angular.io/api/core/Directive), [ViewContainerRef](https://angular.io/api/core/ViewContainerRef) } from '@angular/core';

@[Directive](https://angular.io/api/core/Directive)({

selector: '[ad-host]',

})

export class AdDirective {

constructor(public viewContainerRef: [ViewContainerRef](https://angular.io/api/core/ViewContainerRef)) { }

}

AdDirective injects [ViewContainerRef](https://angular.io/api/core/ViewContainerRef) to gain access to the view container of the element that will host the dynamically added component.

In the @[Directive](https://angular.io/api/core/Directive) decorator, notice the selector name, ad-host; that's what you use to apply the directive to the element. The next section shows you how.

## Loading components

Most of the ad banner implementation is in ad-banner.component.ts. To keep things simple in this example, the HTML is in the @[Component](https://angular.io/api/core/Component) decorator's template property as a template string.

The <ng-template> element is where you apply the directive you just made. To apply the AdDirective, recall the selector from ad.directive.ts, ad-host. Apply that to <ng-template> without the square brackets. Now Angular knows where to dynamically load components.

src/app/ad-banner.component.ts (template)

content\_copytemplate: `

<div class="ad-banner-example">

<h3>Advertisements</h3>

<ng-template ad-host></ng-template>

</div>

`

The <ng-template> element is a good choice for dynamic components because it doesn't render any additional output.

## Resolving components

Take a closer look at the methods in ad-banner.component.ts.

AdBannerComponent takes an array of AdItem objects as input, which ultimately comes from AdService. AdItem objects specify the type of component to load and any data to bind to the component.AdService returns the actual ads making up the ad campaign.

Passing an array of components to AdBannerComponent allows for a dynamic list of ads without static elements in the template.

With its getAds() method, AdBannerComponent cycles through the array of AdItems and loads a new component every 3 seconds by calling loadComponent().

src/app/ad-banner.component.ts (excerpt)

content\_copyexport class AdBannerComponent implements [OnInit](https://angular.io/api/core/OnInit), [OnDestroy](https://angular.io/api/core/OnDestroy) {

@[Input](https://angular.io/api/core/Input)() ads: AdItem[];

currentAdIndex = -1;

@[ViewChild](https://angular.io/api/core/ViewChild)(AdDirective, {[static](https://angular.io/api/upgrade/static): true}) adHost: AdDirective;

interval: any;

constructor(private componentFactoryResolver: [ComponentFactoryResolver](https://angular.io/api/core/ComponentFactoryResolver)) { }

ngOnInit() {

this.loadComponent();

this.getAds();

}

ngOnDestroy() {

clearInterval(this.interval);

}

loadComponent() {

this.currentAdIndex = (this.currentAdIndex + 1) % this.ads.length;

const adItem = this.ads[this.currentAdIndex];

const componentFactory = this.componentFactoryResolver.resolveComponentFactory(adItem.component);

const viewContainerRef = this.adHost.viewContainerRef;

viewContainerRef.clear();

const componentRef = viewContainerRef.createComponent(componentFactory);

(<AdComponent>componentRef.instance).data = adItem.data;

}

getAds() {

this.interval = setInterval(() => {

this.loadComponent();

}, 3000);

}

}

The loadComponent() method is doing a lot of the heavy lifting here. Take it step by step. First, it picks an ad.

**How**loadComponent()**chooses an ad**

The loadComponent() method chooses an ad using some math.

First, it sets the currentAdIndex by taking whatever it currently is plus one, dividing that by the length of the AdItem array, and using the remainder as the new currentAdIndex value. Then, it uses that value to select an adItem from the array.

After loadComponent() selects an ad, it uses [ComponentFactoryResolver](https://angular.io/api/core/ComponentFactoryResolver) to resolve a [ComponentFactory](https://angular.io/api/core/ComponentFactory) for each specific component. The [ComponentFactory](https://angular.io/api/core/ComponentFactory) then creates an instance of each component.

Next, you're targeting the viewContainerRef that exists on this specific instance of the component. How do you know it's this specific instance? Because it's referring to adHost and adHost is the directive you set up earlier to tell Angular where to insert dynamic components.

As you may recall, AdDirective injects [ViewContainerRef](https://angular.io/api/core/ViewContainerRef) into its constructor. This is how the directive accesses the element that you want to use to host the dynamic component.

To add the component to the template, you call createComponent() on [ViewContainerRef](https://angular.io/api/core/ViewContainerRef).

The createComponent() method returns a reference to the loaded component. Use that reference to interact with the component by assigning to its properties or calling its methods.

#### Selector references

Generally, the Angular compiler generates a [ComponentFactory](https://angular.io/api/core/ComponentFactory) for any component referenced in a template. However, there are no selector references in the templates for dynamically loaded components since they load at runtime.

To ensure that the compiler still generates a factory, add dynamically loaded components to the [NgModule](https://angular.io/api/core/NgModule)'s entryComponents array:

src/app/app.module.ts (entry components)

content\_copyentryComponents: [ HeroJobAdComponent, HeroProfileComponent ],

## The AdComponent interface

In the ad banner, all components implement a common AdComponent interface to standardize the API for passing data to the components.

Here are two sample components and the AdComponent interface for reference:

hero-job-ad.component.ts

hero-profile.component.ts

ad.component.ts

content\_copyimport { [Component](https://angular.io/api/core/Component), [Input](https://angular.io/api/core/Input) } from '@angular/core';

import { AdComponent } from './ad.component';

@[Component](https://angular.io/api/core/Component)({

template: `

<div class="job-ad">

<h4>{{data.headline}}</h4>

{{data.body}}

</div>

`

})

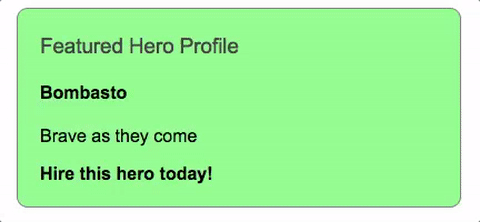
export class HeroJobAdComponent implements AdComponent {

@[Input](https://angular.io/api/core/Input)() data: any;

}

## Final ad banner

The final ad banner looks like this:



See the [live example](https://angular.io/generated/live-examples/dynamic-component-loader/stackblitz.html) / [download example](https://angular.io/generated/zips/dynamic-component-loader/dynamic-component-loader.zip).