# International Javascript Conference October 2019 | Munich



## Ivy, Angular compiler

## Web Components with Angular Elements

# Creating Custom Structural Directives

Ashnita Bali | Twitter

## Directives

Directives are HTML elements and attributes created by Angular applications and Angular libraries.

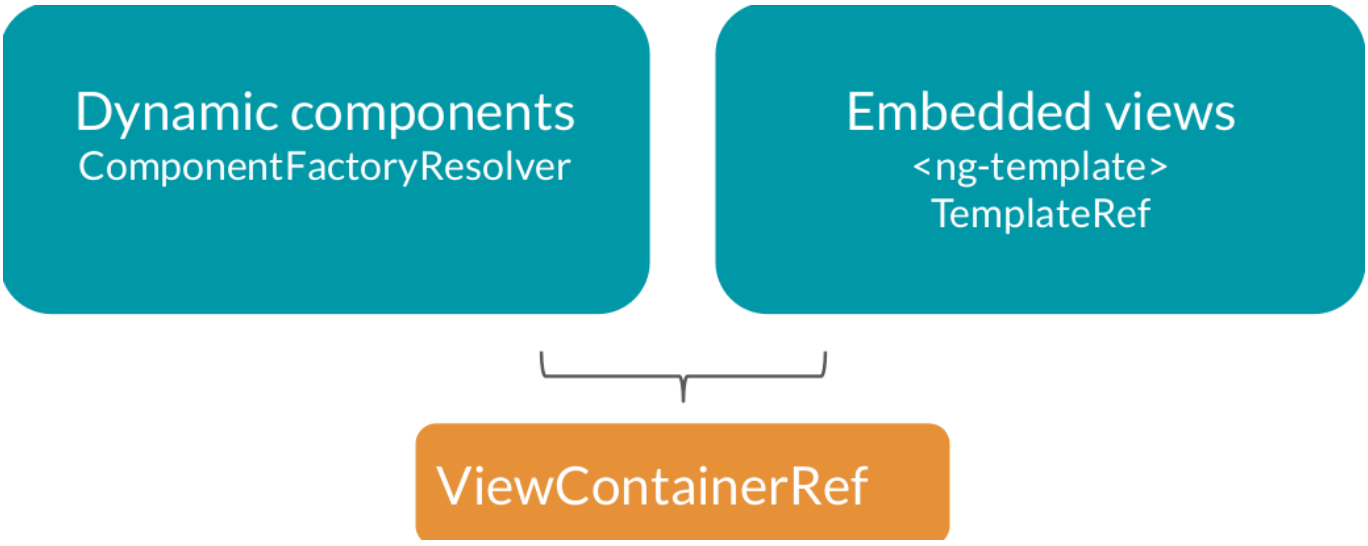
- Components
  - View (html)
  - Styles (css)
  - Data & methods (ts)
- Attribute Directives (Appearance and behaviour) Changes host element's appearance or behaviour
- Structural Directives (Views structure) Changes view structure

**Examples**

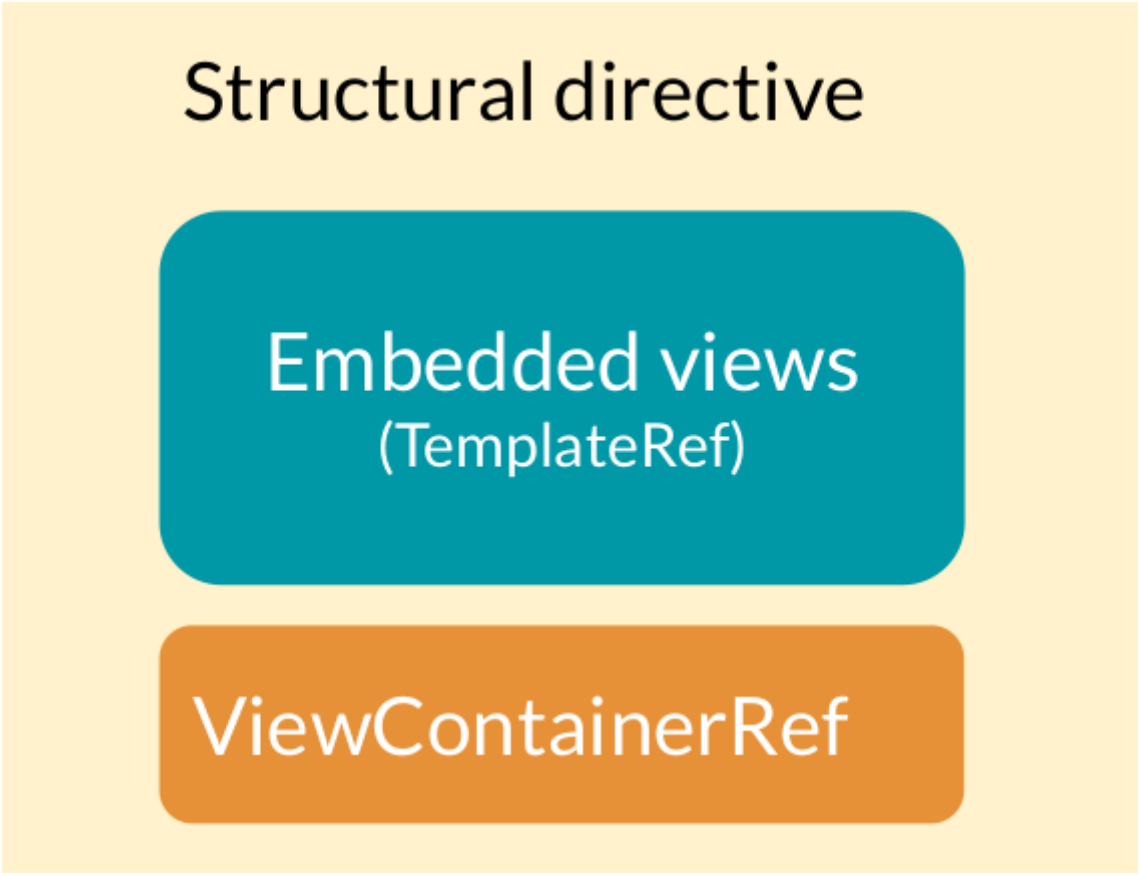
| | Angular Application | Angular Libraries |
|---|---|---|
| **HTML elements** | <app-root><br><app-book-list> | <router-outlet><br><ng-template>, <ng-container>, <ng-content> |
| **HTML attributes** | appCarousel<br>appHighlight | ngFor, ngIf, ngTemplateOutlet, ngSwitchCase<br>routerLink, ngClass, ngStyle, ngForm |

## Views & Dynamic views

A view is the smallest grouping of display elements that can be `created` and `destroyed` together.




Structural directives implement the logic to render embedded views.




Create structural directives when

- rendering a view based on some reusable logic: *appHasRole, *appHasPermission, *appHasFeature
- only implement the functionality, allowing the user to define the view: *appCarousel
- declare variales *appLet

## Steps to create structural directives

**Plan**

- What logic does the structural directive implement?
- API: input properties, context object

```
<div *appHasRole="'admin'">
  User with admin role can see this
</div>
```

1. `ng g d hasRole`

```
  import {Directive} from '@angular/core';
  @Directive({
    selector: '[appHasRole]'
  })
  export class HasRoleDirective {
    constructor() {}
  }
```

2. Get the TemplateRef instance

```
  export class HasRoleDirective {
    constructor(
      private templateRef: TemplateRef<HasRoleContext>
    ) {}
  }
```

```
<ng-template [appHasRole]="'admin'">
  <div>User with admin role can see this</div>
</ng-template>
```

3.Get the ViewContainerRef instance

```
  export class HasRoleDirective {
    constructor(
      private vcr: ViewContainerRef,
      private templateRef: TemplateRef<HasRoleContext>
    ) {}
  }
```

# Are you being servered? Exploring a serverless Web

> Phil Hawksworth | Twitter

- Functions as a service
- Serving websites without webservers
- jamstack, Javascript / API / Markup

**jamstack**
A modern architecture. Create fast and secure sites and dynamic apps with Javascript, APIs, and pre-rendered Markup, served without web servers.

**static sites**

- https://reactjs.org/
- https://yarnpkg.com/lang/en/
- https://vuejs.org/
- https://justdoit.nike.com/

**a little more "dynamic" sites**

- https://www.smashingmagazine.com/
- https://squoosh.app/
- https://www.hawksworx.com/

**illustrating points**

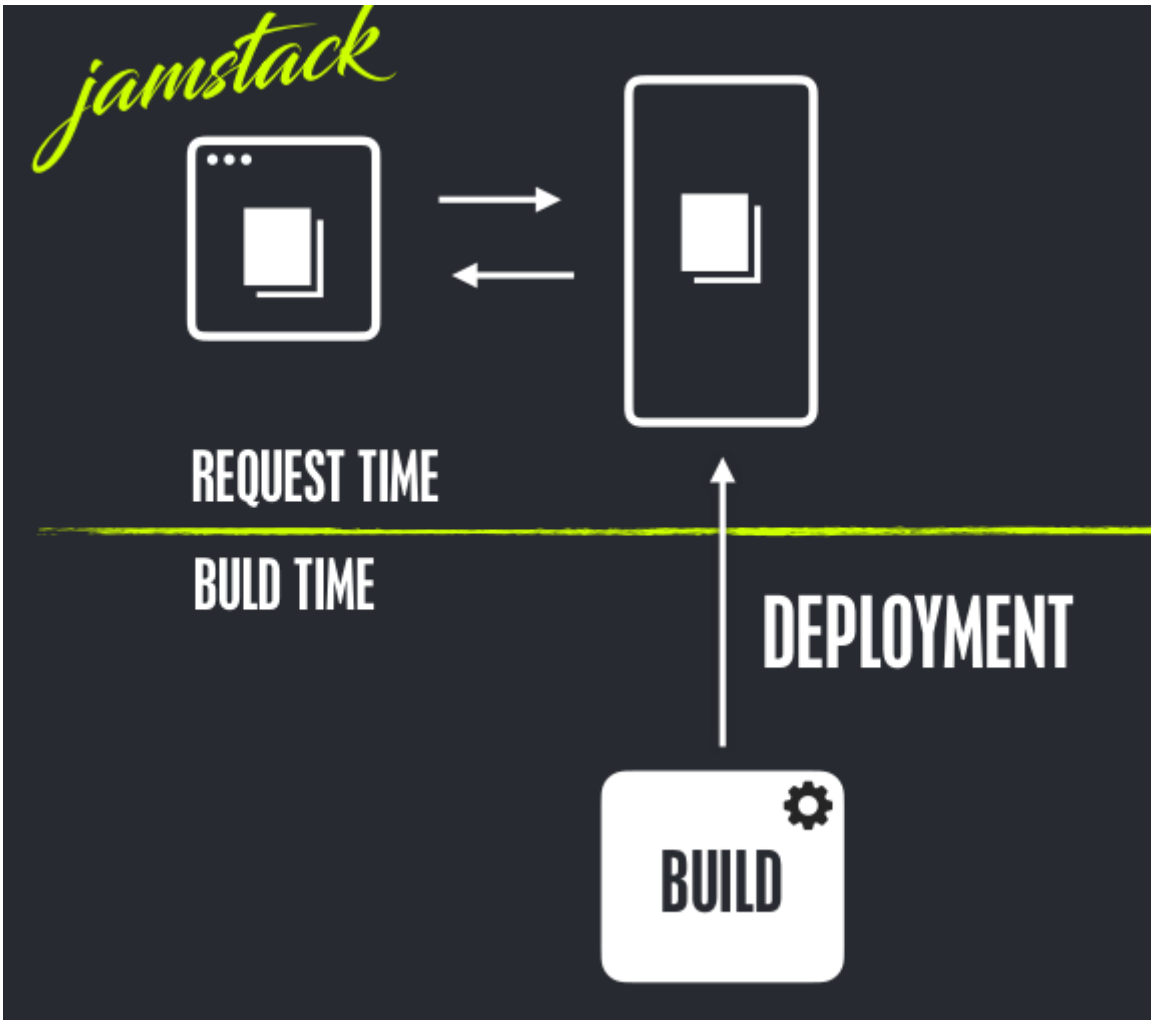
- very low deployment friction
- events and automation

**2 principles**

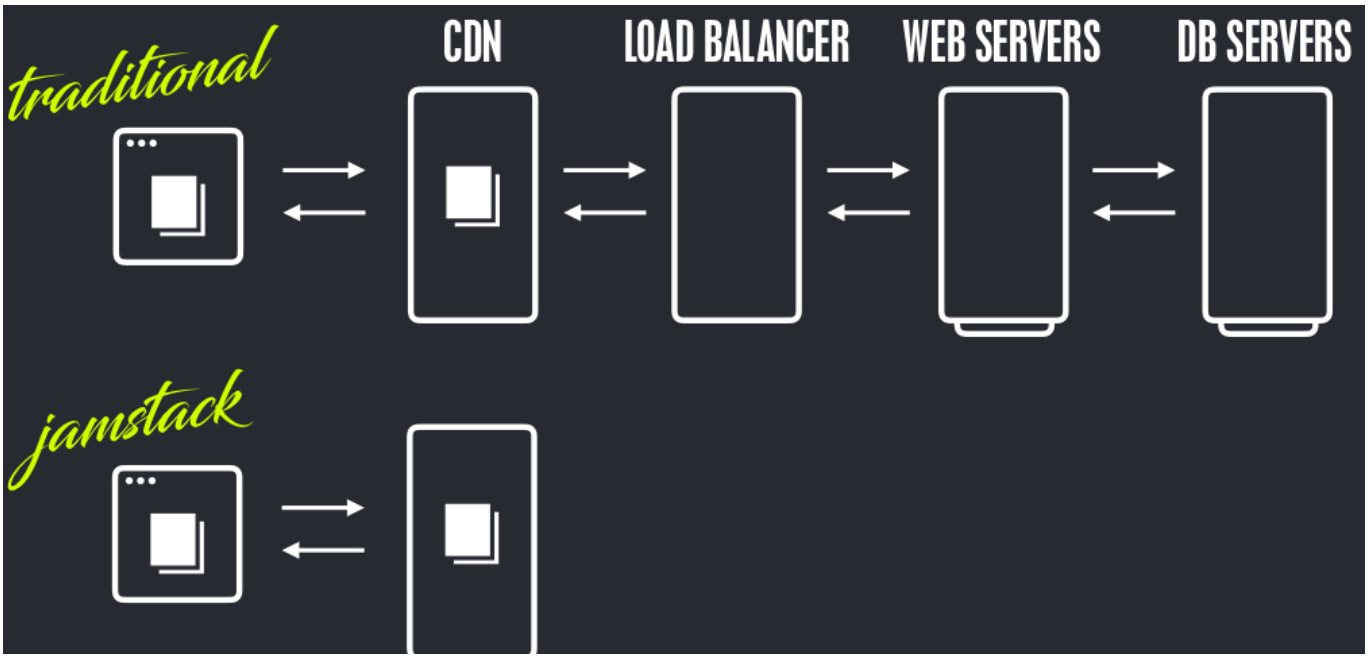
- pre-rendering
- serverless rendering

## PRE-RENDERING

- doing the work now, so your servers don't have to later
- put distance between the complexity and the user




**Jamstack Advantages**

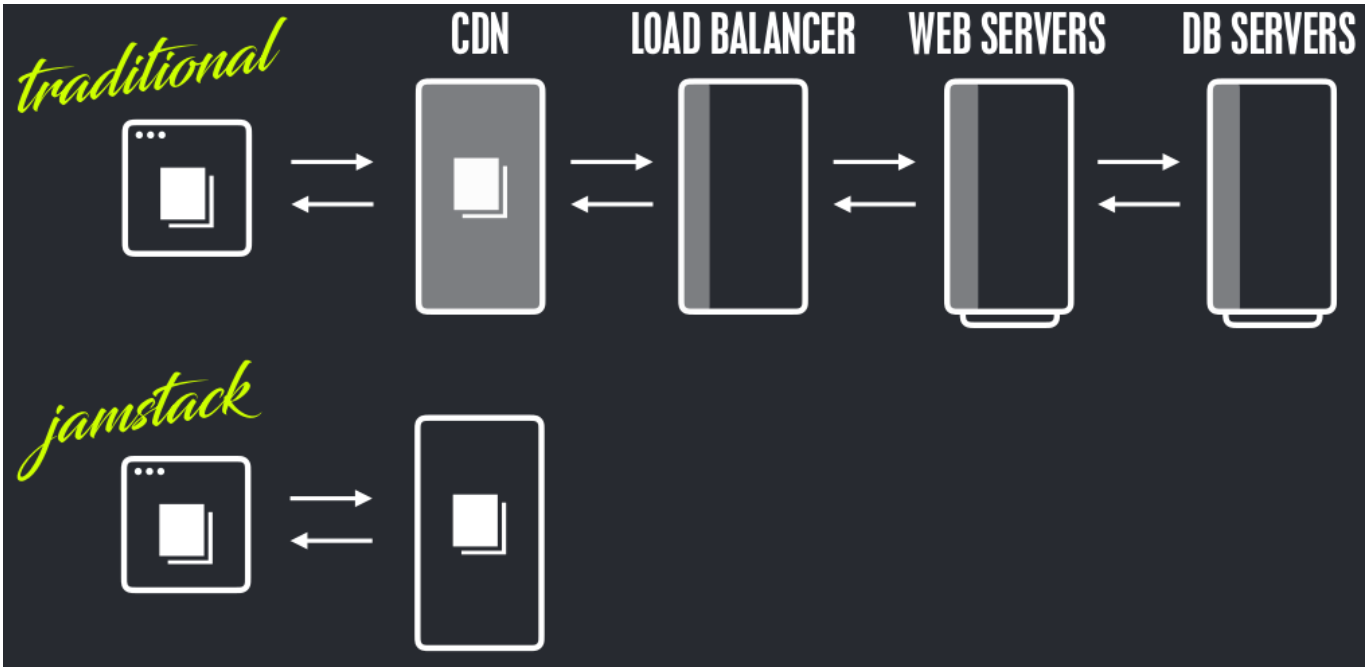



**SECURITY**

- a greatly reduced surface area
- far fewer moving parts to attack

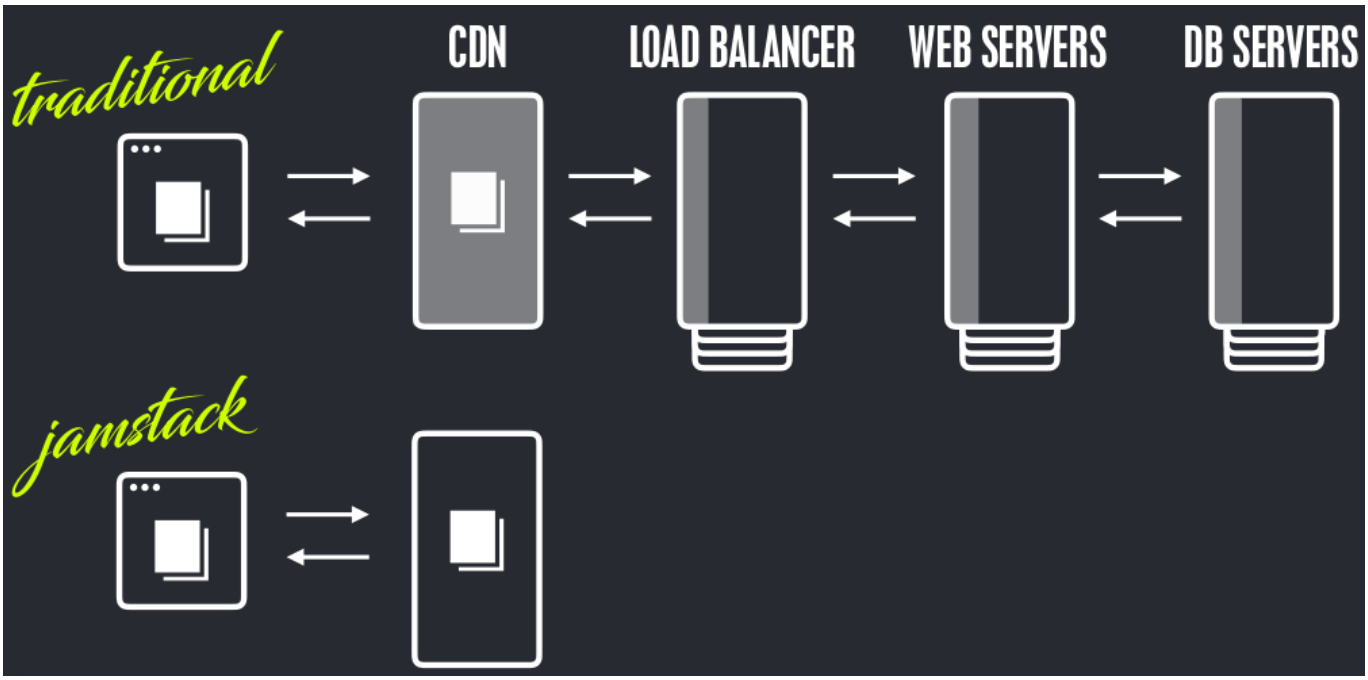
**PERFORMANCE**

Traditional stacks add static layers in order to improve performance. Caching galore




**SCALE**

Traditional stacks add infrastructure in order to scale




## SERVERLESS RENDERING

free from infrastructure but not free from dynamic logic

- augmentation
- enhancement
- static first

example: https://vlolly.net/lolly/3hmtt2r3b/

**PRE-GENERATED PAGES WITH REAL URLS** (EVEN FOR THE USER GENERATED CONTENT)

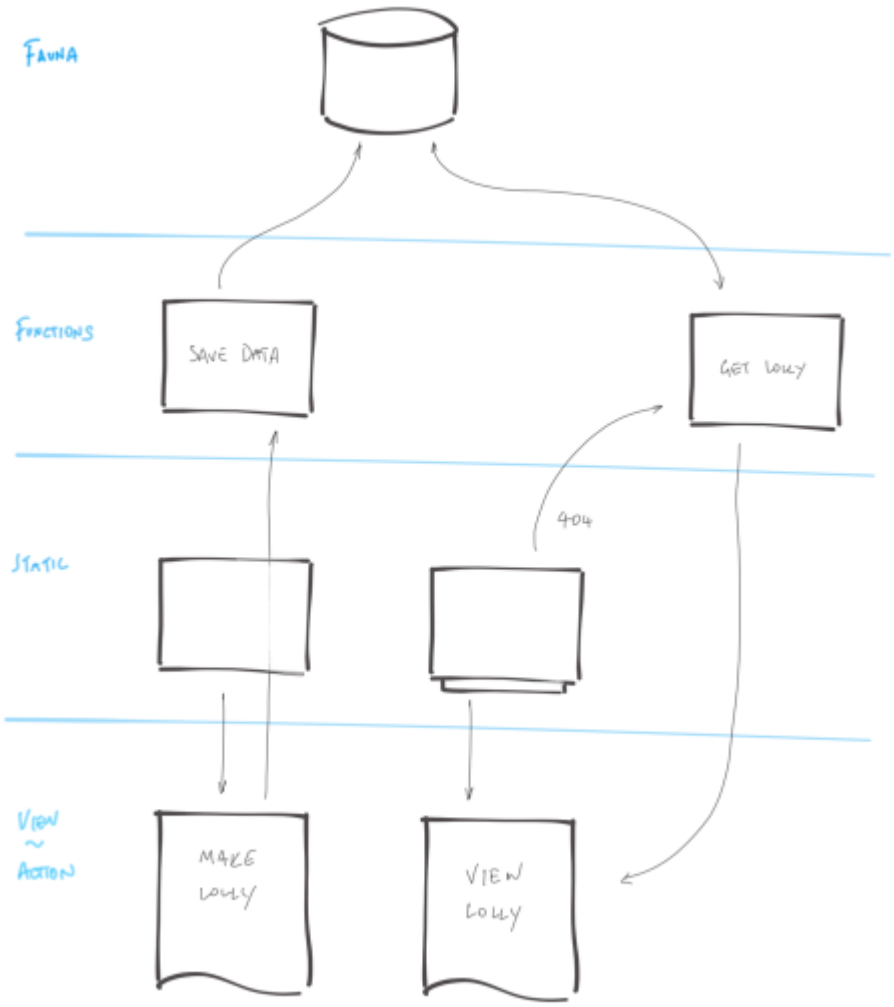
Tool: ELEVENTY

**DATA STORED IN A DATABASE** (BUT DON'T MAKE ME A DBA)

Tool: FAUNADB

**INSTANT ACCESS TO NEW CONTENT** (WITHOUT WAITING FOR A REBUILD)

Tool: NETLIFY FUNCTIONS



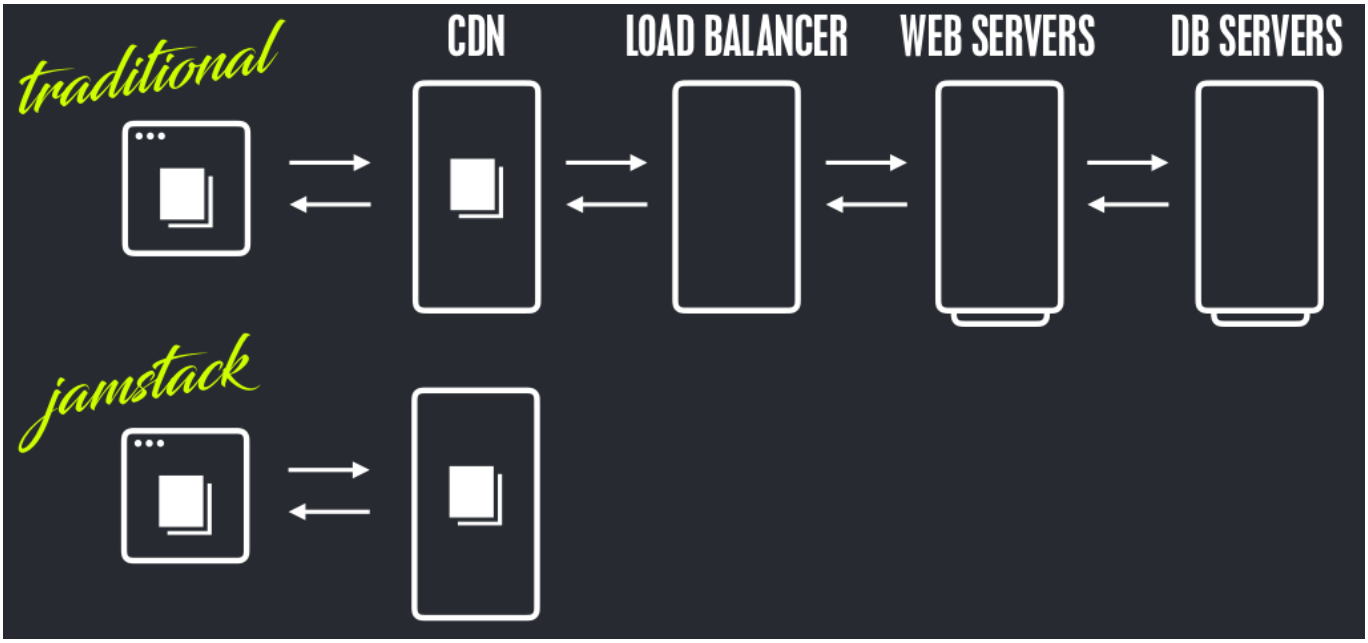


more at https://css-tricks.com/static-first-pre-generated-jamstack-sites-with-serverless-rendering-as-a-fallback/

**enablers**

- custom 404 routing
- events, triggers and automation
- database as a service
- functions as a service

**From** `seeking to optimize by adding static layers` **To** `static first enhancing if required`




Simplifying is not dumbing down It let us focus on what really is important Easier to reason about

## Resources

- Static site generators, https://www.staticgen.com/
- Netlify, https://www.netlify.com/
- various related resources from chris coyier, https://serverless.css-tricks.com/
- citrix presentation from jamstackconf nyc, https://www.youtube.com/watch?v=kvS5h5domf0
- Jamstack conference, https://jamstackconf.com/
- headless / decoupled cms, https://headlesscms.org/
- About jamstack, https://jamstack.org/
- building a url shortener with netlify redirects, https://www.hawksworx.com/blog/find-that-at/
- so i guess we're full stack now?, https://www.youtube.com/watch?v=lFOfQsi5ye0
- modern web development on the jamstack, https://www.netlify.com/oreilly-jamstack/
- jamstack comments example, https://jamstack-comments.netlify.com/
- jamstack slack, https://jamstack.slack.com

# Profiling Javascript apps like a pro

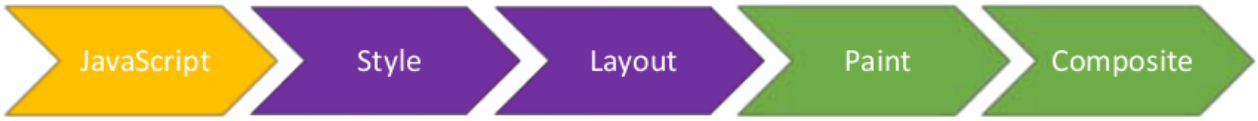
> Gil Fink | Twitter

## Understanding Browsers

### Refresh Rates

- Devices refresh their screens 60 times a second = 60 fps.
- That means that each frame should take 16ms since 1 second / 60 = 16.66 ms.
- In reality a frame takes ~ 10 ms due to browsers management overhead.

### Pixel pipeline




- **JavaScript**
  used to handle work that will result in visual changes. CSS Animations, Transitions, and the Web Animations API are also used here.
- **Style**
  browser figures out which CSS rules should be applied to elements based on CSS selectors. The style is then calculated to each element.
- **Layout**
  the browser calculates how much space each element takes up and where it is on screen. Each element affects other elements in the layout, Web layout model.
- **Paint**
  the browser paints all the pixels on screen. It draws every visual part of an element (text, color, images and etc)
- **Composite**
  the browser draws the elements according to their layer, if elements overlap each other. Happens on the machine GPU, therefore this step is very fast.

### Reflows

Reflow might occurs whenever a visual change requires a change in the layout of the page. Examples: browser resize, DOM manipulation and etc. All the flow of the process (pixel pipeline) will run again

### Repaints

Repaint occurs when a visual change doesn't require recalculation of the whole layout. Examples: element visibility change, changes in text color or background colors and etc. All the flow of the process (pixel pipeline) except *Layout* will run again.


> Reflows/Repaints
>
> Try to minimize them as much as possible

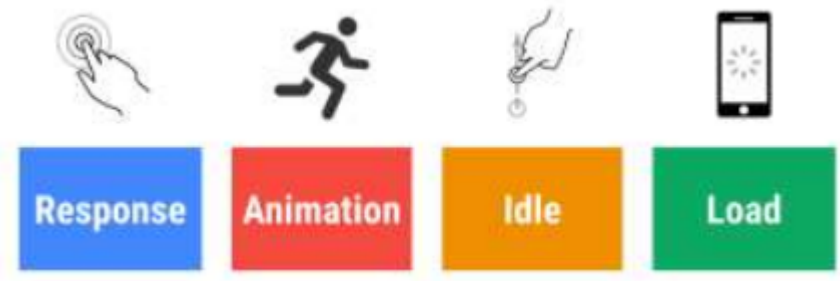
**Changes without Reflow or Repaint**

JavaScript or CSS changes that don't affect neither layout or paint. The flow of the process (pixel pipeline) will run again without layout and paint.

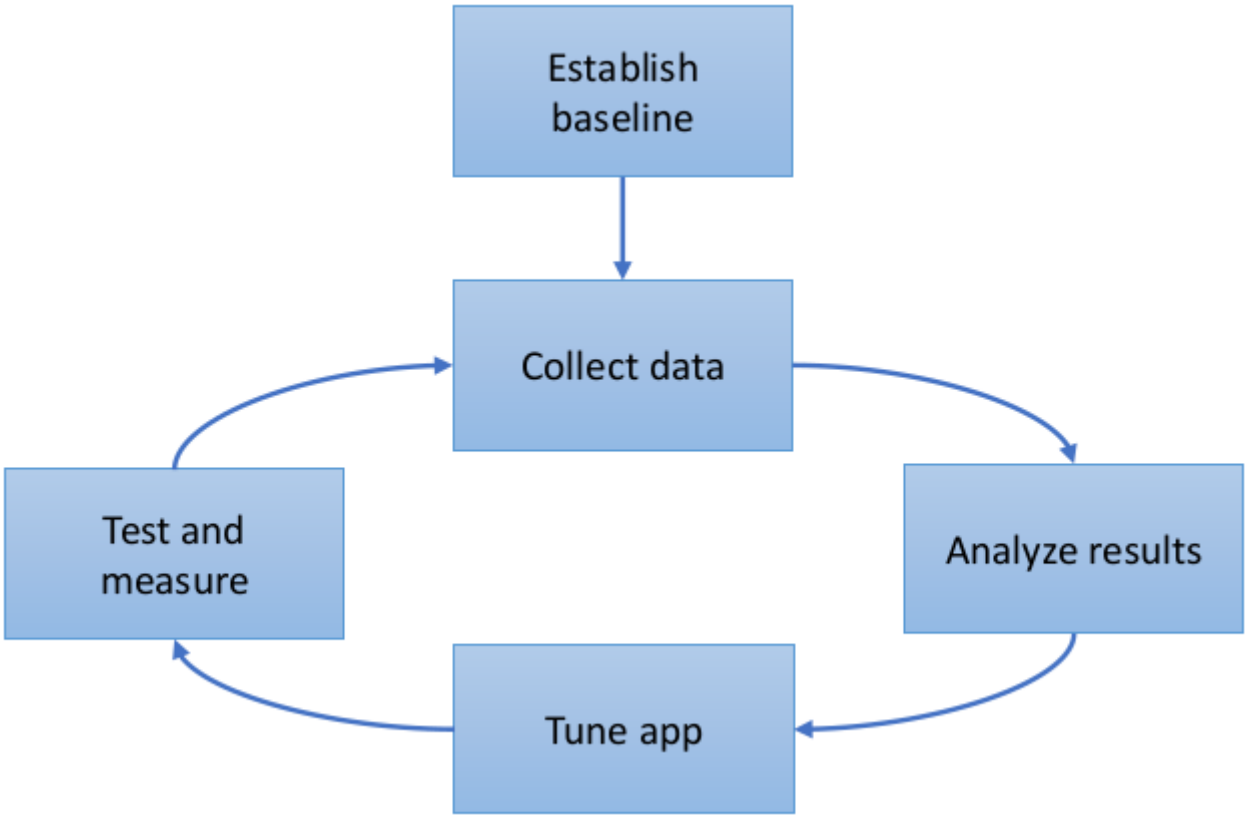

## RAIL Model

User-centric performance model

- **Response**
  process events in under 50 ms
- **Animation**
  produce a frame in 10 ms
- **Idle**
  maximize idle time
- **Load**
  deliver content and become interactive in under 5 secs




## Profiling Process




## Memory Leaks

Memory that isn't required by an app but isn't returned to the pool of free memory

**Type of Common Javascript Leaks**

- Accidental global variables
  easily solved with *use strict* or using linters
- Forgotten timers or callbacks
- Out of DOM references
- Closures

## Angular Performance Tuning Tips

- OnPush Change Detection
- Lazy Loading Modules
- Preserve Whitespaces
- Enable Production Mode
- ngFor Track By Function
- Avoid Function Calls in Views/Avoid Getters in Views
- Use Pure Pipes
  if the input always produce the same output the pipe can be marked as pure
- Use Async Pipes
  async pipes automatically handle all observable cleanup