

Entorno React.js

Patricio López

patricio@lopezjuri.com

github.com/mrpatiwi

[telegram.me/mrpatiwi](https://t.me/mrpatiwi)

Contenidos

- Javascript y ECMAScript
- Node.js
- React.js
- Arquitecturas posibles
- Mi experiencia
- DEMO
- Referencias y links

Javascript

- Lenguaje interpretado
- Tipado dinámico y débil
- Permite programación:
 - Funcional
 - Orientada a Prototipos
 - Reactiva
- Naturalmente asíncrono
- Sin threads (es single-thread)

A large, dark blue, stylized 'JS' logo is centered within a yellow square. The letters are bold and modern, with the 'J' and 'S' having thick strokes and rounded terminals.

Javascript

- No existe una implementación oficial de un intérprete
- Cada navegador implementa la suya



ECMAScript

Fija el estándar del lenguaje Javascript

Versiones de ECMAScript (ES)

- **ES5**: Compatible en todas partes
- **ES6** ó ES2015: Versión moderna
 - Grandes y potentes funcionalidades nuevas
 - Javascript pasa de ser un juguete a un **lenguaje de verdad**
 - **No todos los navegadores la soportan al 100%**
- **ES7** ó ES2016: Versión actual
 - Agrega pocas cosas a ES6

Toda versión nueva soporta todas las versiones anteriores

ES5

```
function Hello(name) {  
  name = name || 'Annon';  
  this.name = name;  
}  
  
Hello.prototype.hello = function hello() {  
  return 'Hello ' + this.name +  
  '!';  
};  
  
Hello.sayHelloAll = function () {  
  return 'Hello everyone!';  
};  
  
var hw = new Hello();  
hw.hello();
```

ES6

```
class Hello {  
  constructor(name = 'Annon') {  
    this.name = name;  
  }  
  
  hello() {  
    return `Hello ${this.name}!`;  
  }  
  
  static sayHelloAll() {  
    return 'Hello everyone!';  
  }  
}  
  
const hw = new  
Hello('Patricio');  
hw.hello();
```

ES7+

Funcionalidades y especificaciones propuestas van pasando por etapas hasta que finalmente son implementadas.

ES7+ apunta a las que están muy avanzadas en el proceso, pero no oficialmente incluidas.

Javascript fuera del navegador



Node.js

- Toma V8 de Chrome
- Hace los bindings al sistema nativo
- Single Thread
- I/O non-blocking
 - Toda llamada web o a disco la hace de manera asíncrona
- La *versión 6* soporta el ~95% del estándar **ES6**

Ejemplo con Express.js

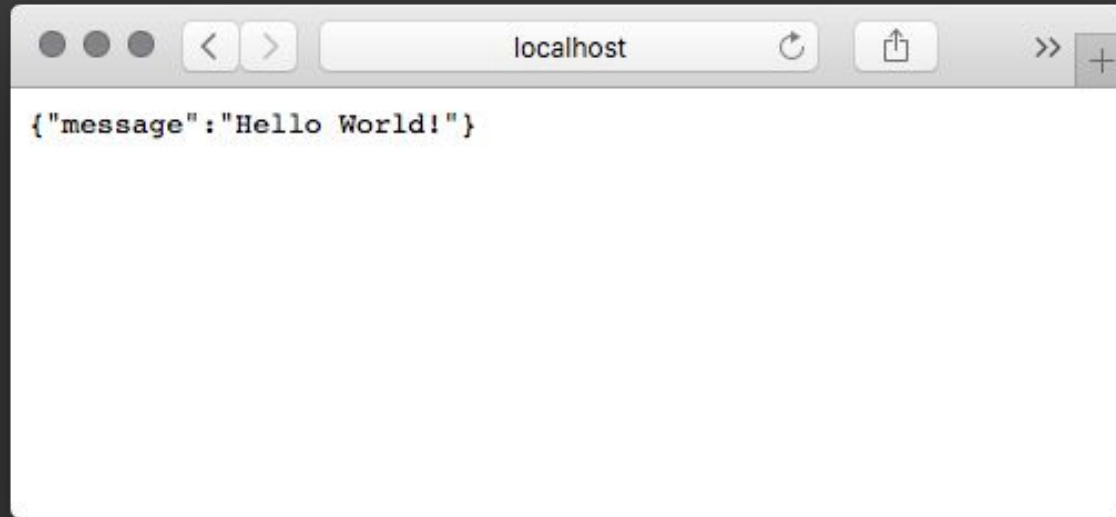
```
const express = require('express');
const app = express();

app.get('/', (req, res) => {
  res.send({ message: 'Hello World!' });
});

const PORT = 3000;

app.listen(PORT, () => {
  console.log(`Example app listening on port ${PORT}!`);
});
```

Ejemplo con Express.js



¿Librerías? Módulos



Así como *pip* de Python y las *gemas* de Ruby

¿Entonces qué podemos hacer?

Javascript en el servidor y computador con **Node.js**:

- Simples scripts

```
$ node mi-script.js
```

- Líneas de comando para la terminal

```
$ mi-app iniciar --name cli
```

- Podemos hacer el servidor de aplicaciones web



METEOR
express

¿Entonces qué podemos hacer?

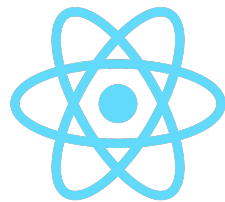
Javascript en el cliente:

- Páginas web simples
- SPA: Single Page Application:
 - Javascript tiene toda la lógica para generar y mover HTML en el cliente
 - El servidor solo envía JSON
 - El servidor ya no se preocupa de generar vistas
 - Todo es más rápido y fluido
- Aplicaciones de escritorio multiplataforma
- Aplicaciones móviles multiplataforma

¿Entonces qué podemos hacer?

Javascript en el cliente:

- Páginas web simples
- SPA: Single Page Application:
 - Javascript tiene toda la lógica para generar y mover HTML en el cliente
 - El servidor solo envía JSON
 - El servidor ya no se preocupa de generar vistas
 - Todo es más rápido y fluido
- Aplicaciones de escritorio multiplataforma
- Aplicaciones móviles multiplataforma



React.js



- Framework (parece más una librería) para hacer interfaces
 - Obviamente en Javascript
- Mantenido por Facebook
- Lo más *'hot'* del momento por todas las cosas que se pueden hacer
- Alto performance
- Server-side rendering
 - Podemos pre-generar la vista en el servidor con Node.js



A screenshot of a web browser displaying the React.js live editor on the website `facebook.github.io`. The browser's address bar shows the URL and standard navigation icons. The website's header includes the React logo, navigation links for "Docs", "Support", "Download", and "Blog", a search bar labeled "Search docs...", and links to "GitHub" and "React Native".

The main content area is divided into two sections. On the left, under the "Live JSX Editor" tab, is a code editor with a light yellow background containing the following JavaScript code:

```
'use strict';

class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello {this.props.name}</h1>
        <p>This is like HTML </p>
      </div>
    );
  }
}

ReactDOM.render(<HelloMessage name="John" />, mountNode);
```

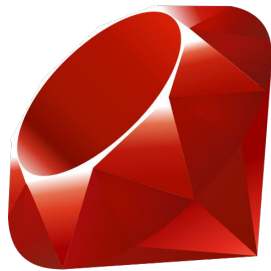
On the right, under the "Compiled JS" tab, is a preview window with a white background. It displays the rendered output of the code: a large heading "Hello John" followed by a paragraph "This is like HTML".

No todos los módulos en **npm** son
compatibles simultáneamente en
Node.js y *Browsers*



Arquitectura de la aplicación

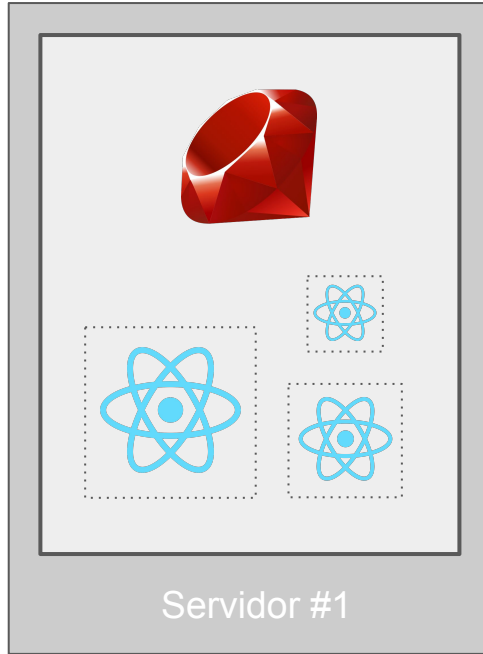
Tenemos 3 opciones



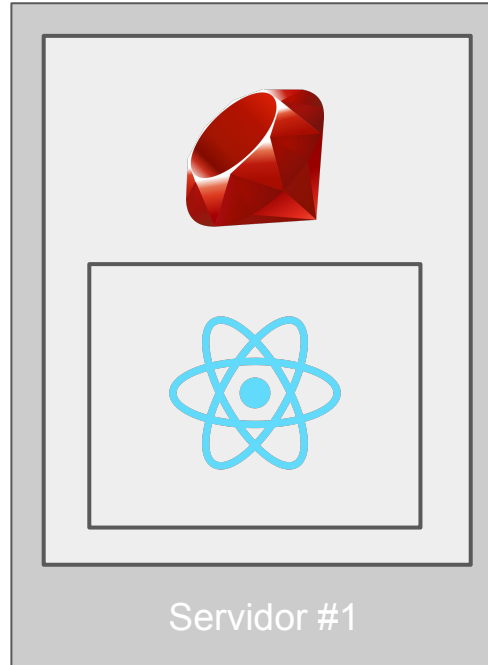
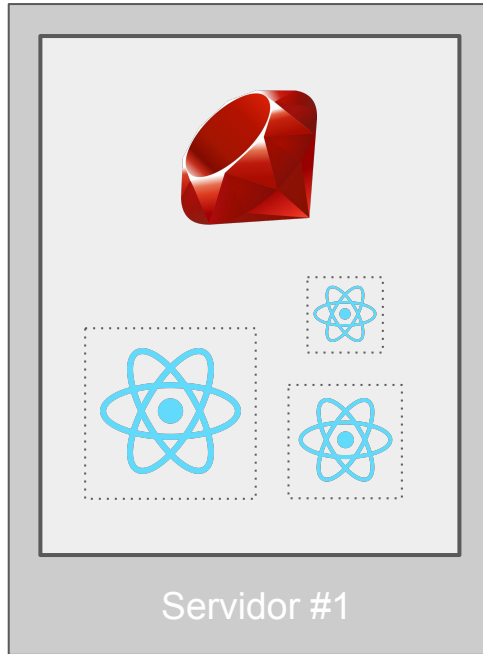
A MODO DE EJEMPLO

Usaré Rails para referirme al servidor
Puede ser cualquier Framework

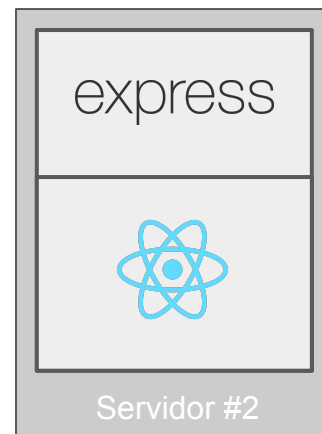
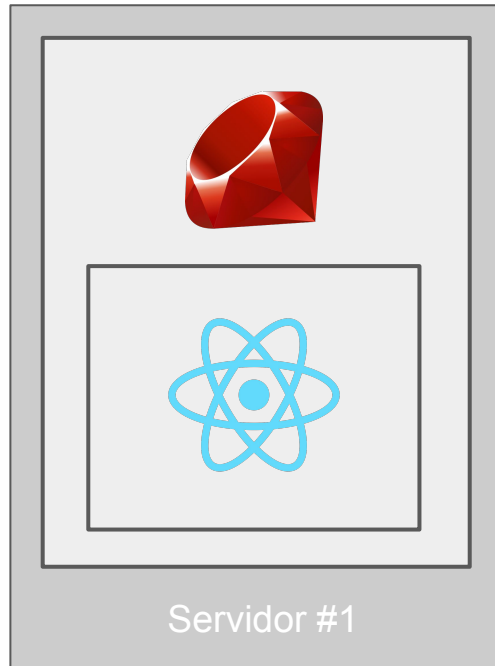
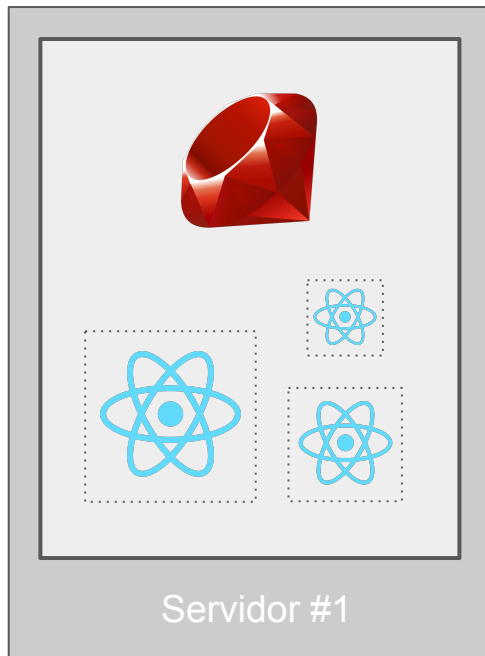
Opción 1: Usar React.js como *template-engine*



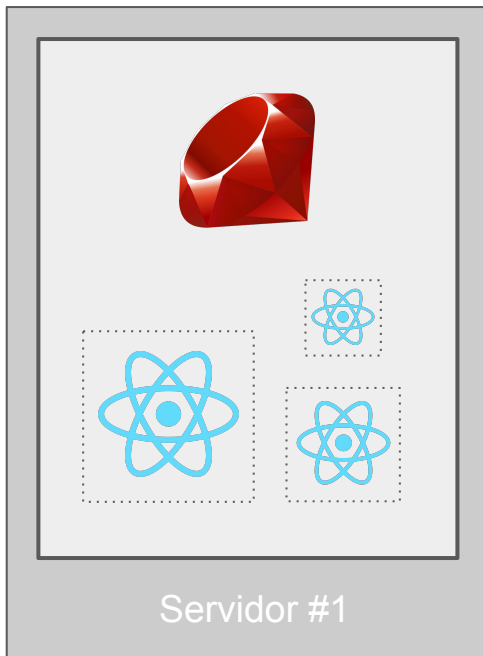
Opción 2: Aplicación React.js servida ahí mismo



Opción 3: Aplicación en React en su propio server



Opción 1: Usar React.js como *template-engine*



Usamos React.js **solo para generar ciertos componentes que necesitan mucha interacción con el usuario**

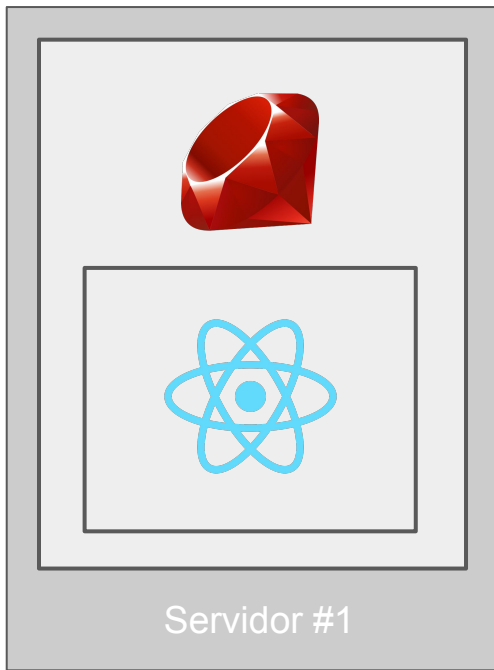
PROS:

- Mantenemos la lógica compleja en Rails
 - Sesión
 - Router
 - No necesariamente tenemos que hacer una API
- Poco uso de React.js y de manera simple

CONS:

- Alto acoplamiento
- Poca compatibilidad con otras librerías
- Muy limitado
- Dependencias de librerías no-oficiales en distintos lenguajes

Opción 2: Aplicación React.js servida ahí mismo



SPA en su totalidad es programada y servida dentro de la misma app en la carpeta *'public'*. Ahora sí necesitamos programar una API.

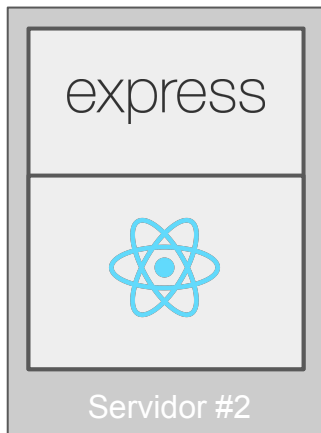
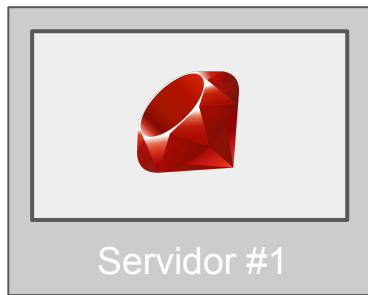
PROS:

- Podemos usar React.js y librerías en su totalidad
- El servidor solo se preocupa de servir JSON y la aplicación en React.js como archivos estáticos.

CONS:

- Trabajo manual
- Ya no podemos hacer 'scaffold'
- Difícil trabajar dos equipos en un mismo repositorio
- No escala

Opción 3: Aplicación en React en su propio server



Ponemos a las aplicaciones (cliente y servidor) de manera que ninguna conozca la implementación de la otra.

(Pueden estar en la misma máquina, pero siguen siendo apps por separado).

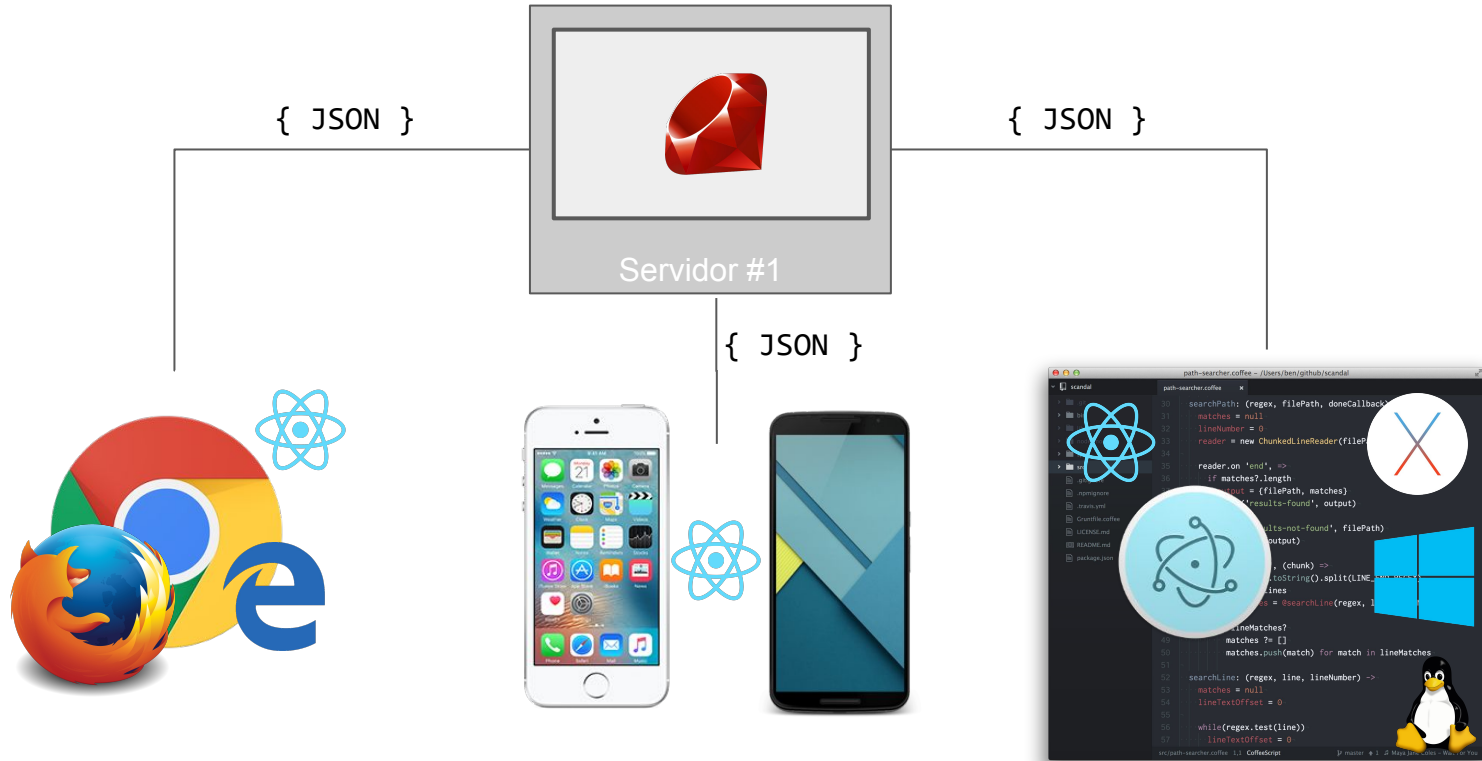
PROS:

- Podemos usar React.js y librerías en su totalidad
- El trabajo en equipo fluido
- Escala

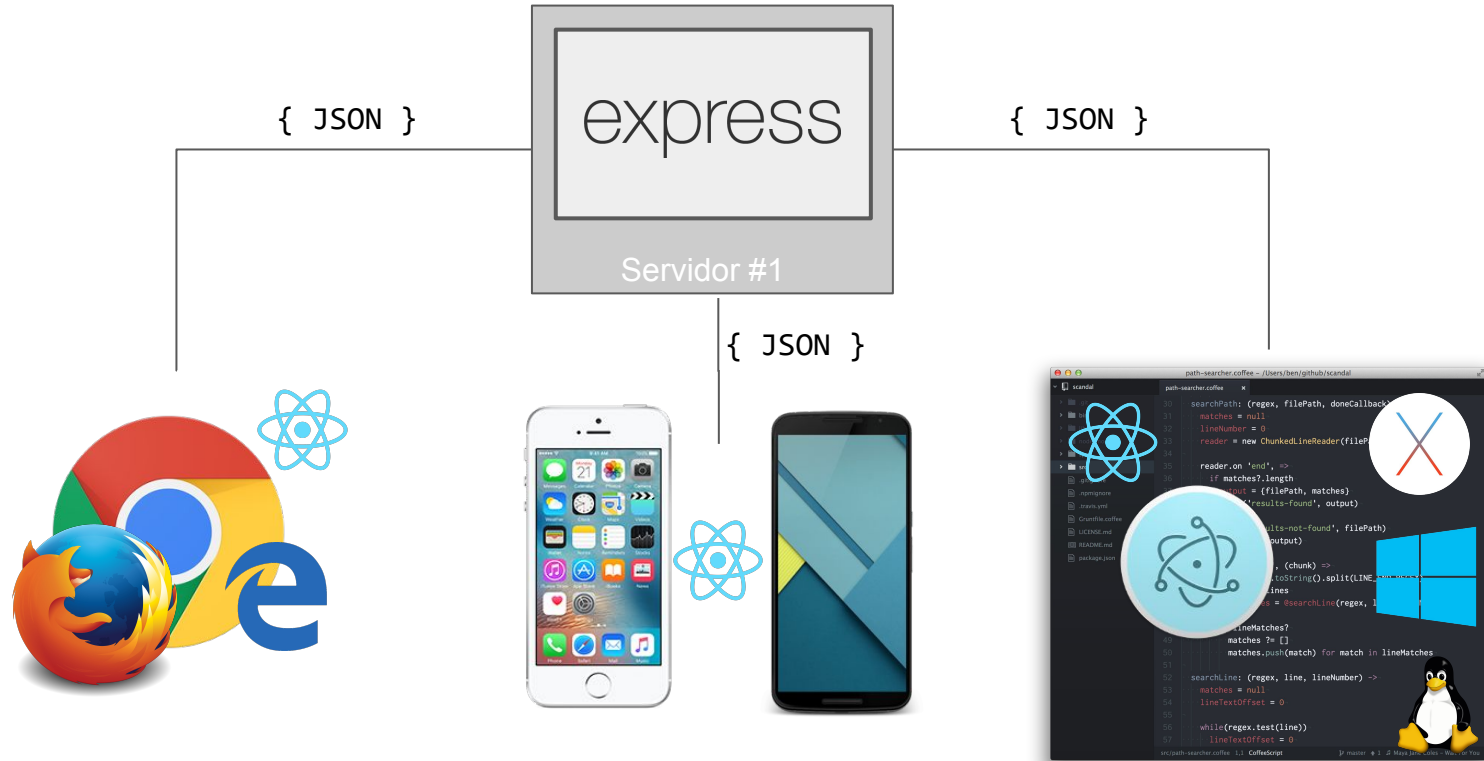
CONS:

- Trabajo manual
- Ya no podemos hacer 'scaffold'
- Requiere otra máquina o del setup de poner ambos en una.

¿Qué eventualmente podríamos llegar a hacer?



Podríamos hacer todo el sistema en un solo lenguaje



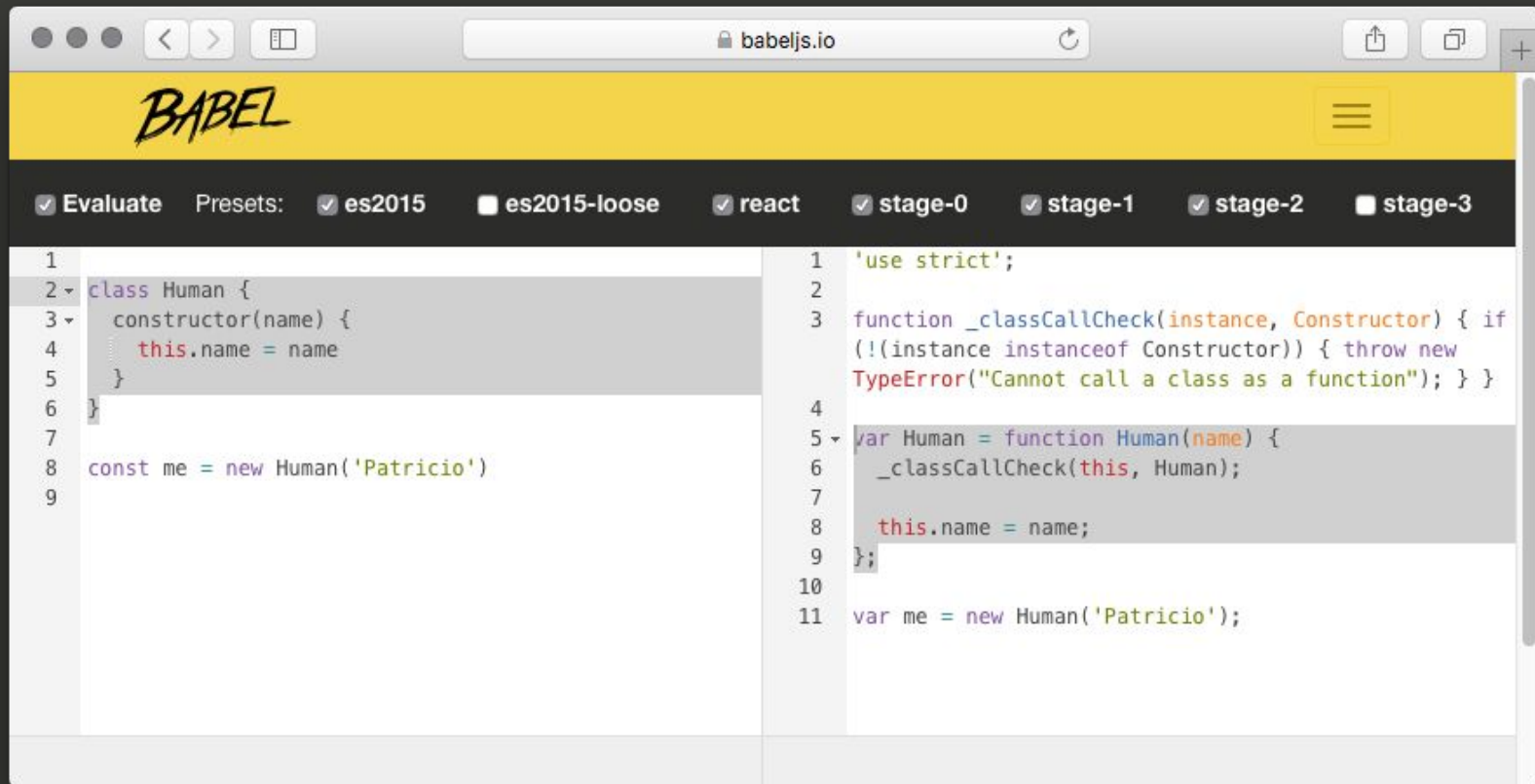
¿Qué versión de ECMAScript uso?

Ni siquiera Node.js soporta completamente ES6

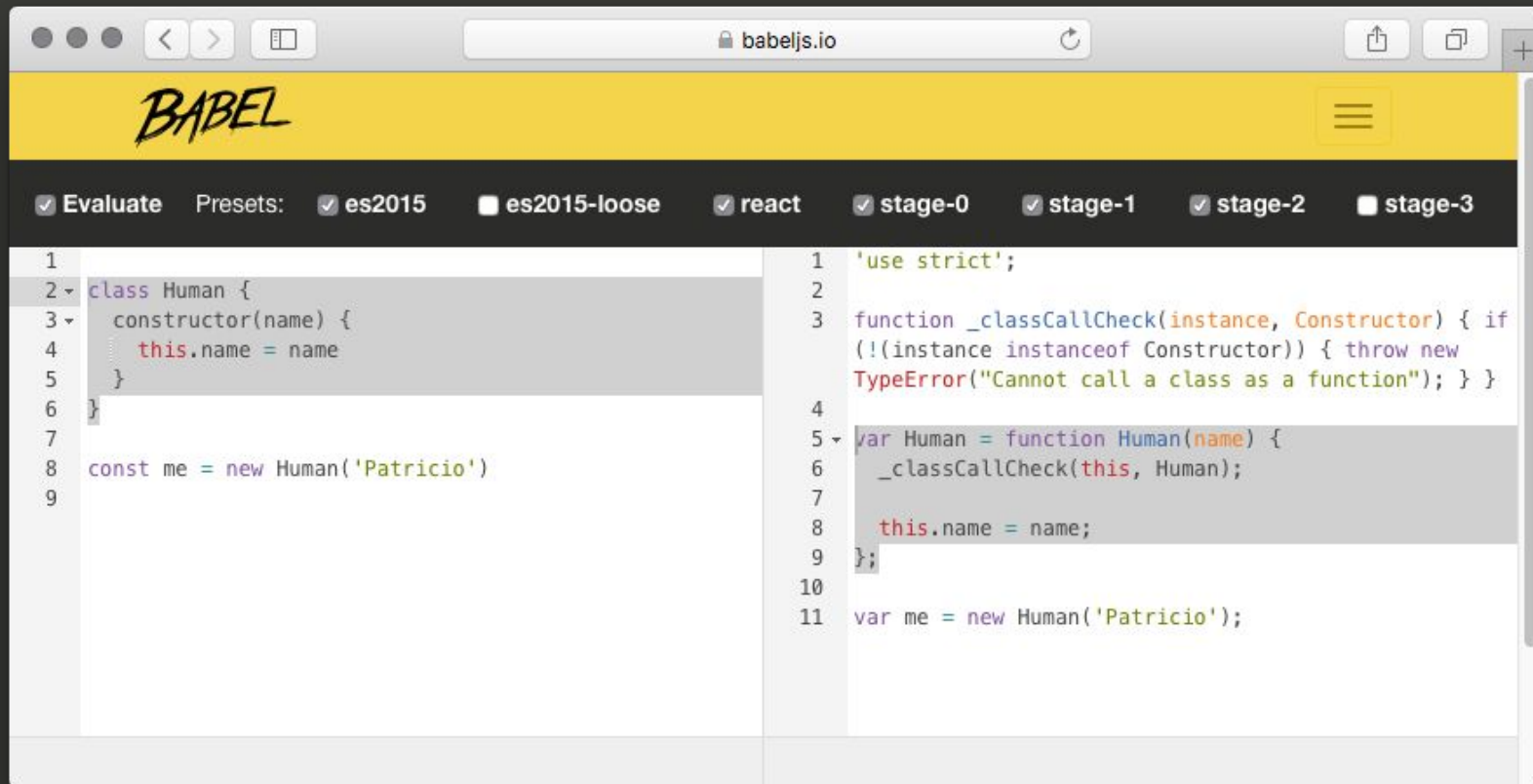
Necesitamos convertir ES7+ a ES5

BABEL

“Transpilador”



“Transpilador”



The screenshot shows the Babel REPL interface in a web browser. The URL is `babeljs.io`. The Babel logo is in the top left. A navigation bar contains the word "Evaluate" and a list of presets: `es2015` (checked), `es2015-loose` (unchecked), `react` (checked), `stage-0` (checked), `stage-1` (checked), `stage-2` (checked), and `stage-3` (unchecked). The interface is split into two panels. The left panel shows the input code:

```
1
2 class Human {
3   constructor(name) {
4     this.name = name
5   }
6 }
7
8 const me = new Human('Patricio')
9
```

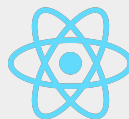
 The right panel shows the output code:

```
1 'use strict';
2
3 function _classCallCheck(instance, Constructor) { if
4   (!instance instanceof Constructor) { throw new
5   TypeError("Cannot call a class as a function"); } }
6
7 var Human = function Human(name) {
8   _classCallCheck(this, Human);
9   this.name = name;
10 };
11
12 var me = new Human('Patricio');
```

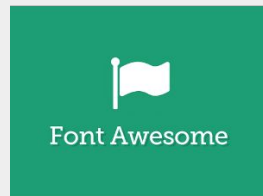
Se instala con npm, esta es solo una demo en vivo

¿...y cómo lo integro todo?

- ¿Qué pasa si uso preprocesadores de CSS como **SCSS** o *Less*?



Bootstrap



¿...y cómo lo integro todo?

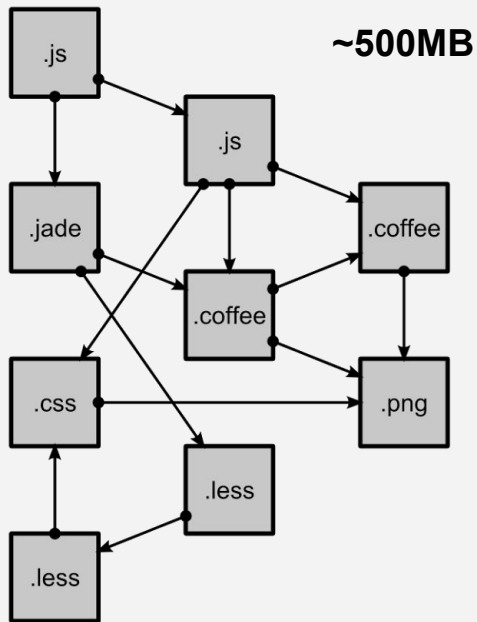
- ¿Qué pasa si uso preprocesadores de CSS como **SCSS** o **Less**?
- También quiero *minificar* y *ofuscar* el código.
- Etc...

Necesitamos una especie de “compilador”

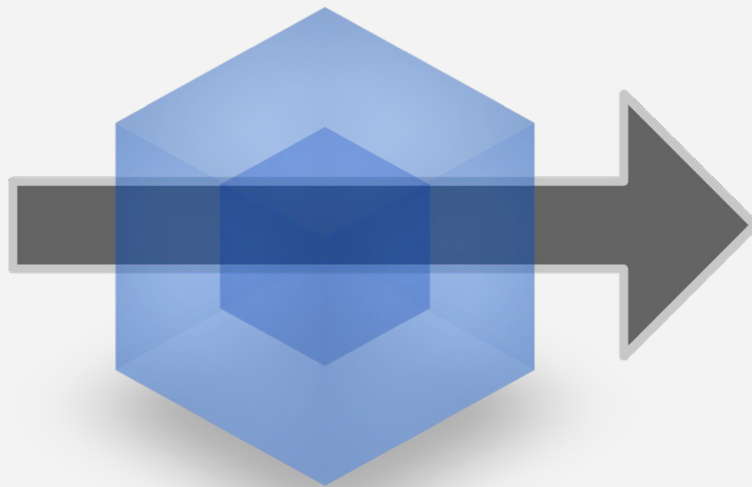


Está pensado para
clientes móviles y web

webpack
MODULE BUNDLER

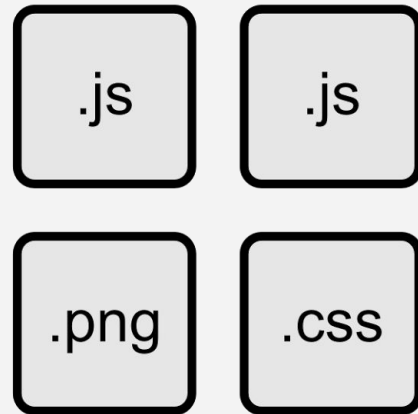


modules
with dependencies



webpack
MODULE BUNDLER

< 2MB



static
assets

Webpack

- Empaqueta el código del frontend
- Remueve código no usado
- Integración con plugins como:
 - Babel
 - SCSS
 - Optimizador de código
 - Etc...

Ufff... ¿Algo más?

Aseguremos la calidad del código



Integración con **Atom**,
Sublime, etc...

Aseguremos la calidad del código

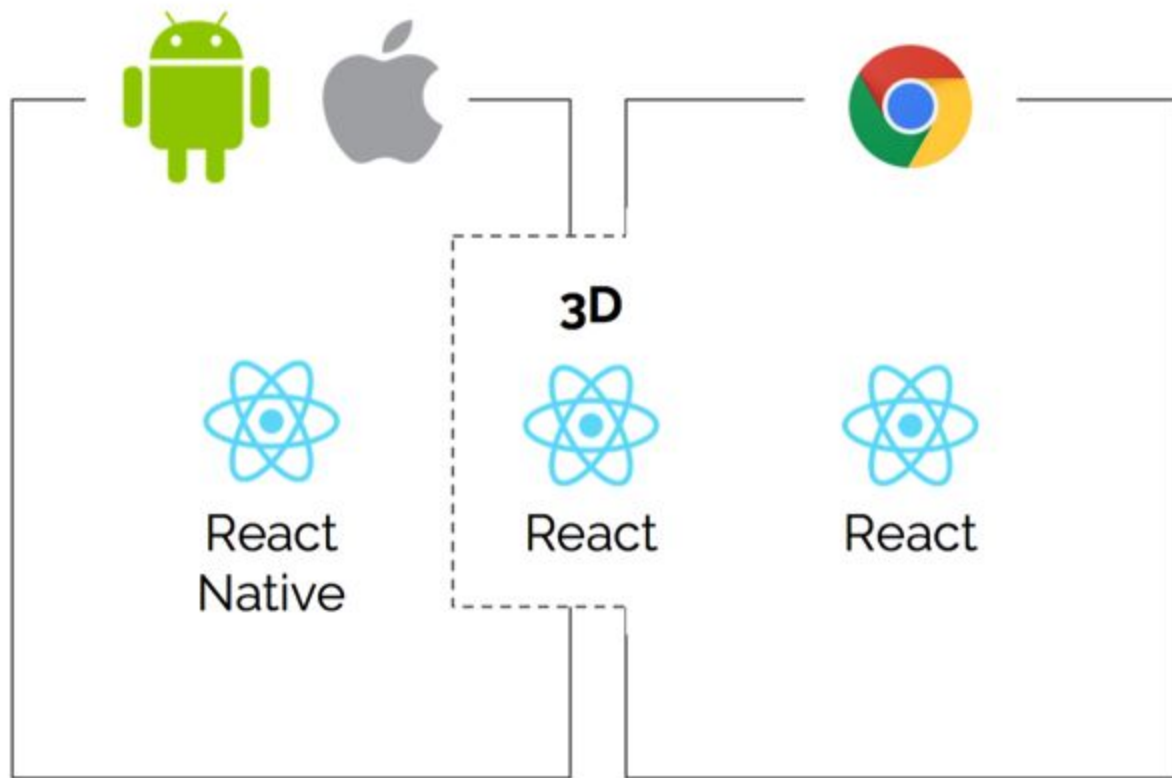


Mi experiencia



Atlas

Contenido enriquecido y evaluaciones



Componente 3D compartido

Web API

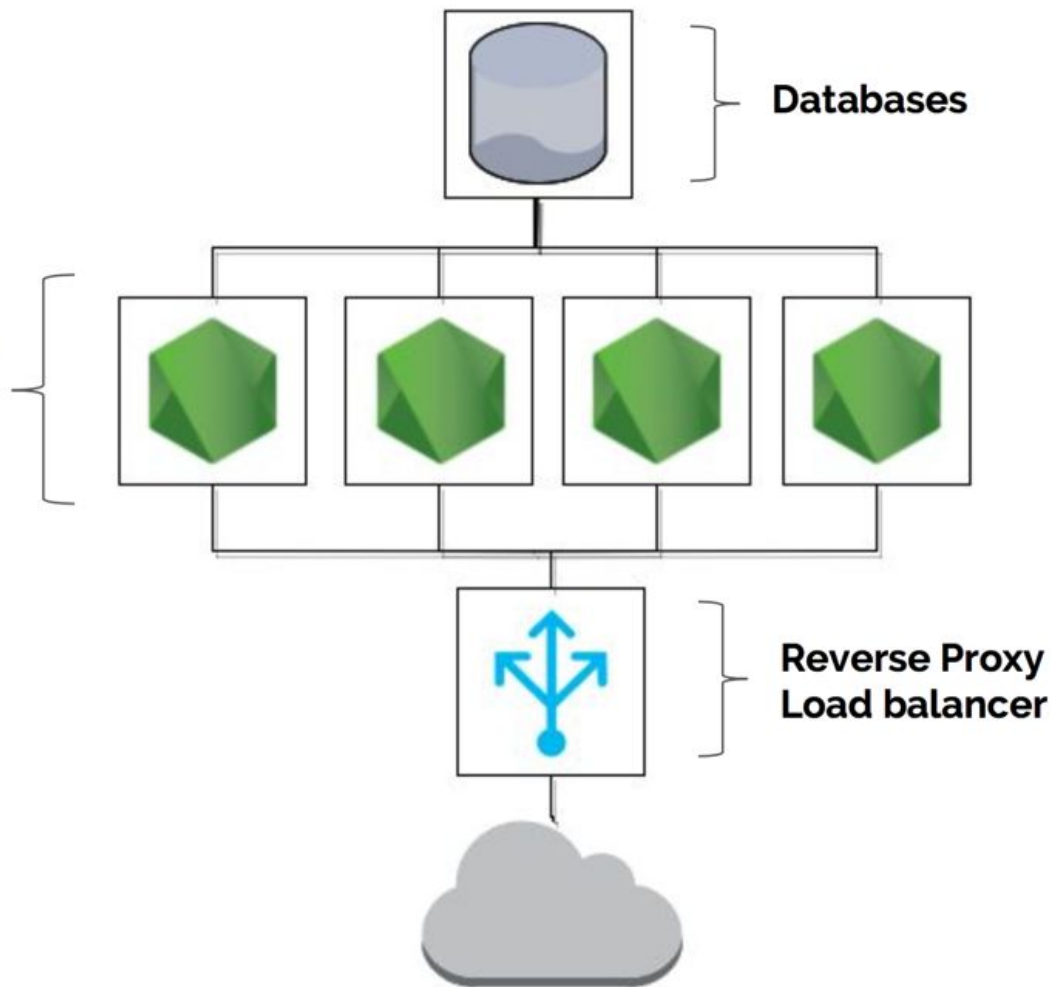
Servidor Ingeniería

REST API
Cluster Node.js

Databases

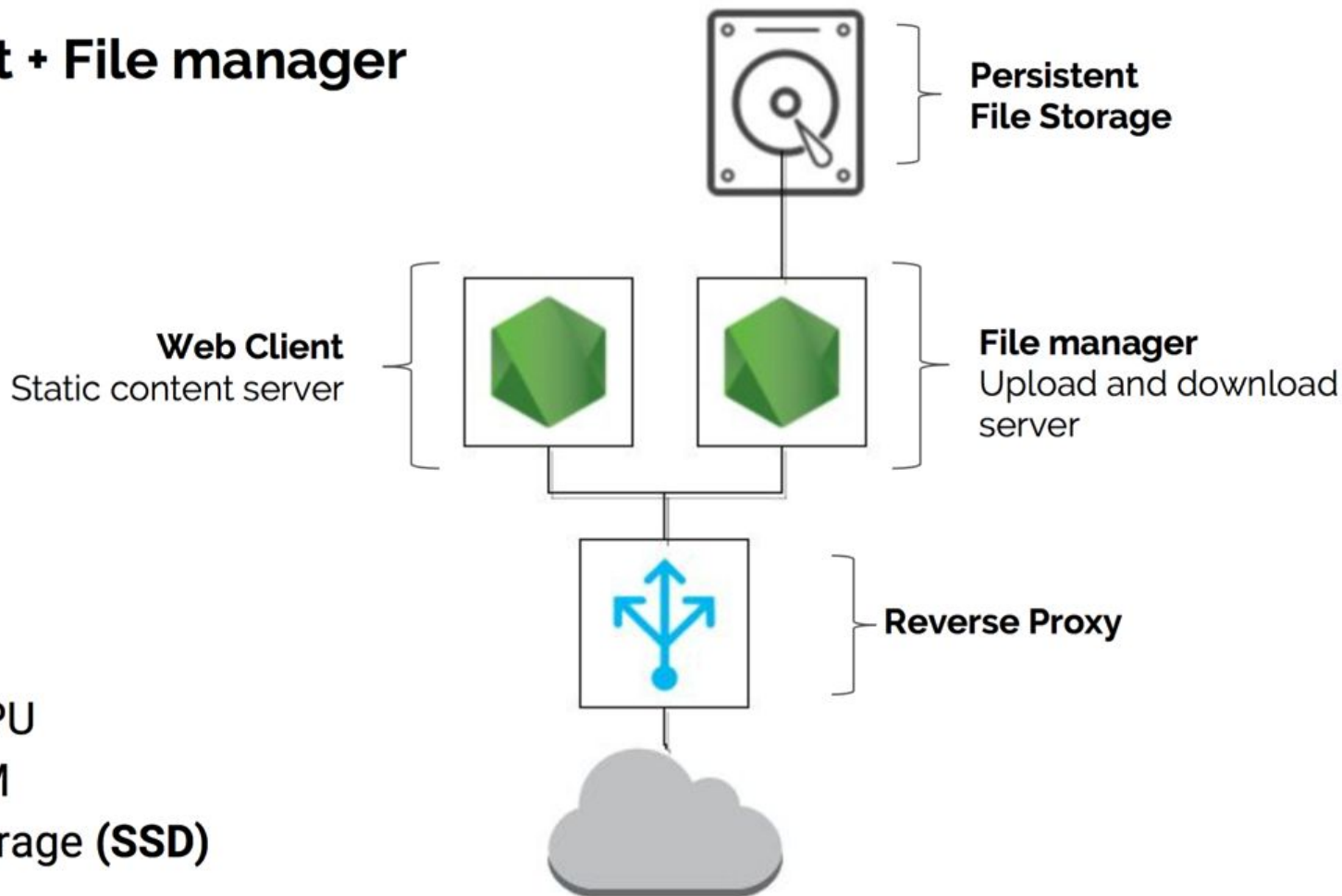
Reverse Proxy
Load balancer

- **4 Core CPU**
- **4 GB RAM**
- **30GB Storage**



Web Client + File manager

Digital Ocean



- 2 Core CPU
- 1 GB RAM
- 30GB Storage (**SSD**)

Lo bueno

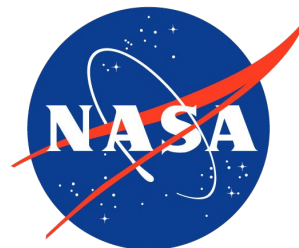
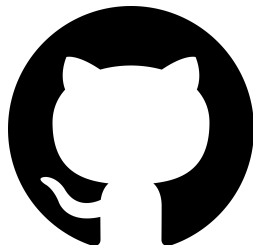
- Solo necesitamos un lenguaje
- Máxima movilidad dentro del equipo
- Fácil creación de interfaces potentes y dinámicas
- Disponibilidad de muchas librerías
 - Muchas de estas son universales (web, react-native, node.js)
- La comunidad más viva en Github y Stackoverflow
- **El equipo aprendió herramientas que siguen usando fuera del ramo**

Lo malo

- Javascript es un lenguaje muy distinto a los usuales
 - **Lo 'básico' se bastante avanzado**
 - Es fácil confundirse si no lo entiendes bien
 - Muchas versiones y cuesta encontrar la mejor fuente de información
 - Es difícil
- Cuesta partir porque hay que hacer muchas cosas a mano
- Mucha dependencia en librerías
 - Pueden existir problemas de compatibilidad
 - Se actualizan muy seguido



NETFLIX



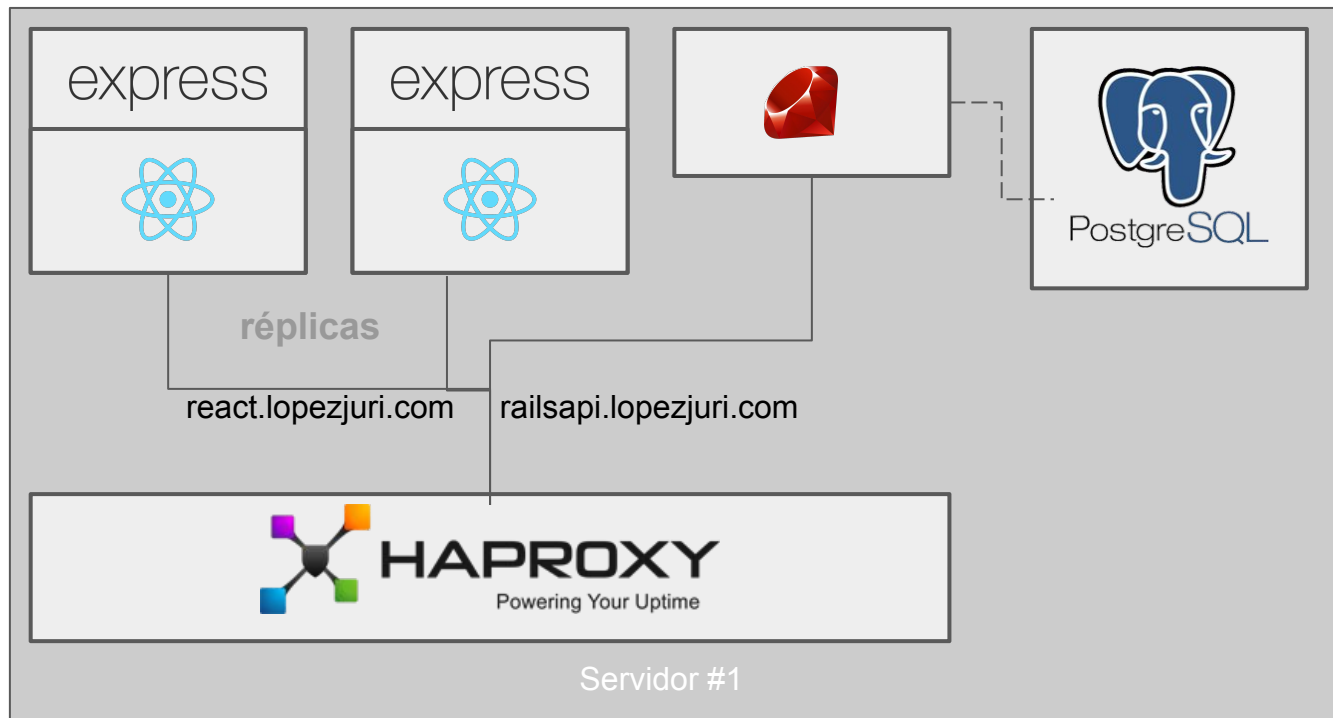
Quienes usan Node.js / Javascript

DEMO: <https://github.com/mrpatiwi/entorno-react>

- En un mismo servidor montaremos:
 - Web API en Express.js en modo de cluster
 - Una aplicación en React.js
 - Un balanceador de carga
- Usaremos Docker y un servidor con Ubuntu
- Configuraremos el DNS para poder acceder a esta

Si tuviéramos más plata podríamos usar más servidores

DEMO: <https://github.com/mrpatiwi/entorno-react>



Referencias y links importantes

- Mejor tutorial para React.js
 - <https://egghead.io/courses/react-fundamentals>
- Mejor scaffold para un app en React.js
 - <https://github.com/facebookincubator/create-react-app>
- Artículo mio de como hacer un buen setup
 - <https://medium.com/@patriciolpezjuri/using-create-react-app-with-react-router-express-js-8fa658bf892d>