

Aplicaciones Distribuidas en Internet

Tema 1:
Introducción a las
Aplicaciones Web. Apps
web “en la nube”

Tema 1: Introducción a las Aplicaciones web. Apps web
“en la nube”

1.1

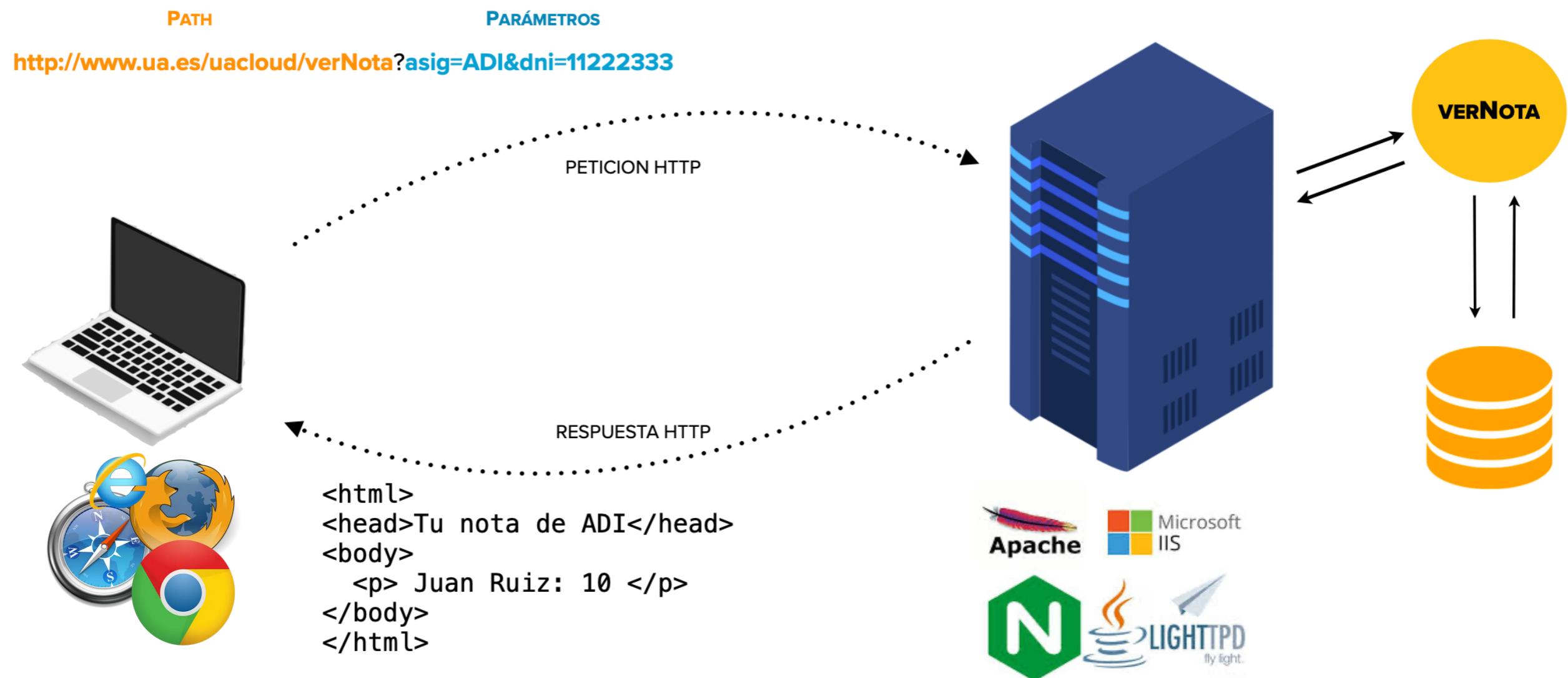
Introducción a las Apps
web

Aplicación Web

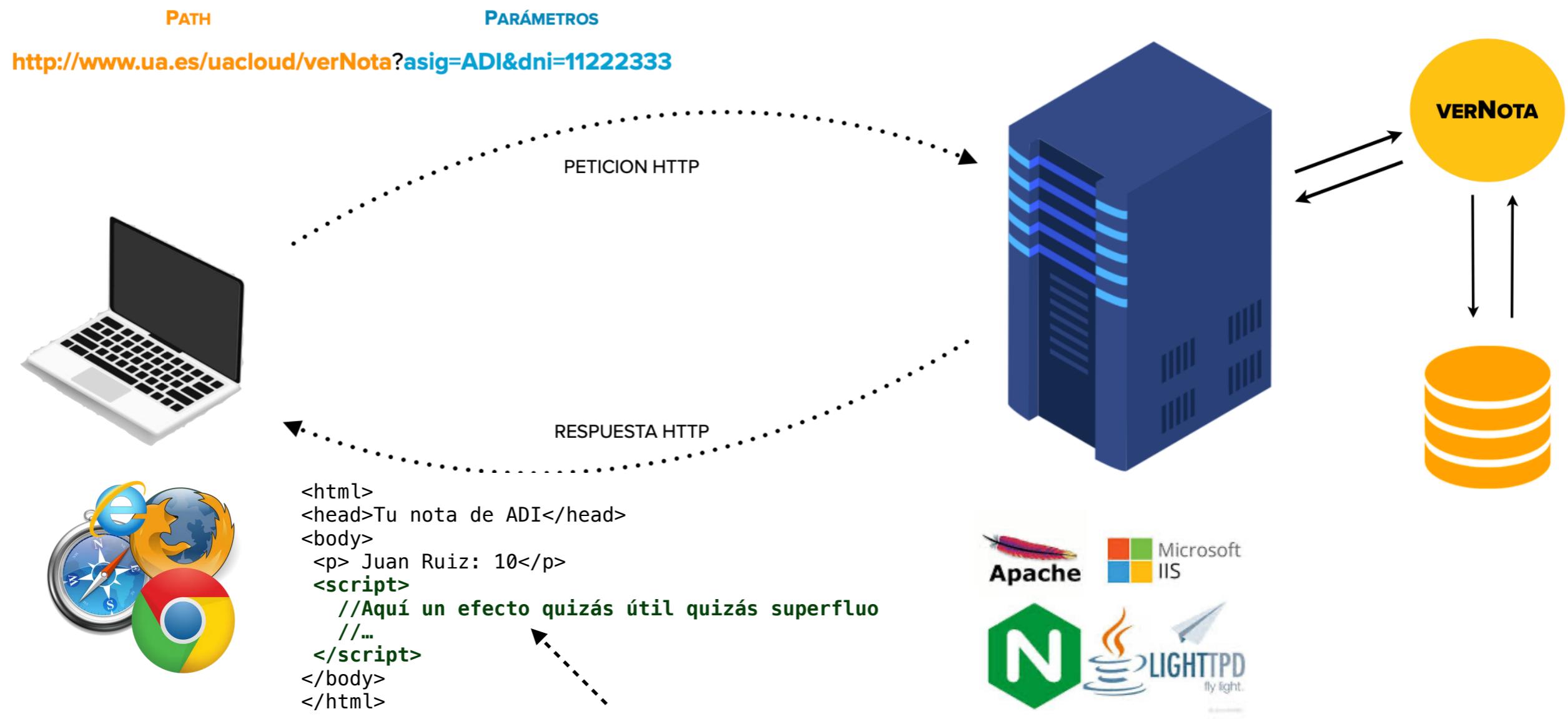
- Aplicaciones cliente/servidor en las que:
 - El **protocolo** de comunicación es **HTTP**
 - El **cliente** es un **navegador web**



En el principio solo había backend



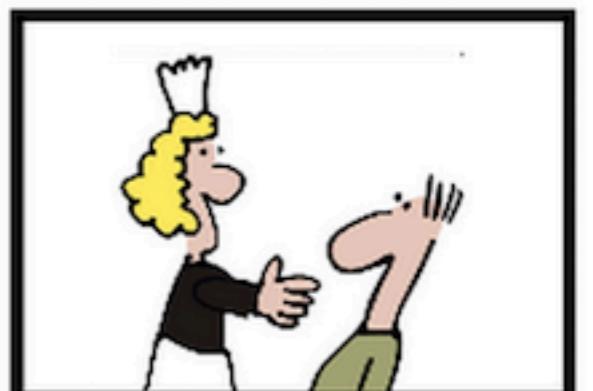
Luego apareció el desarrollo Frontend...



Y apareció AJAX...

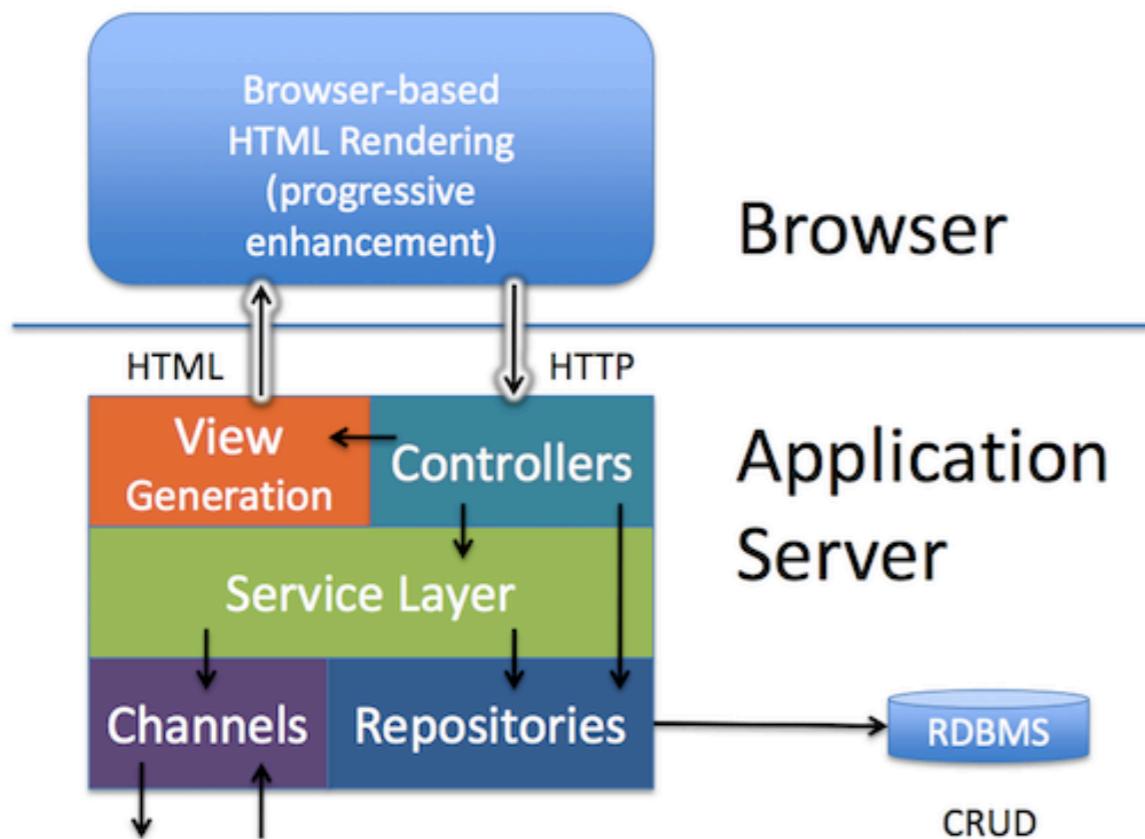
- En la web “tradicional” cuando se hacía una petición HTTP desde el navegador se **cargaba totalmente una nueva página**
- Tras aparecer **AJAX**, se pueden disparar peticiones desde Javascript y **la página no cambia** (con los datos obtenidos podemos cambiar parte de ella)

SIMPLY EXPLAINED

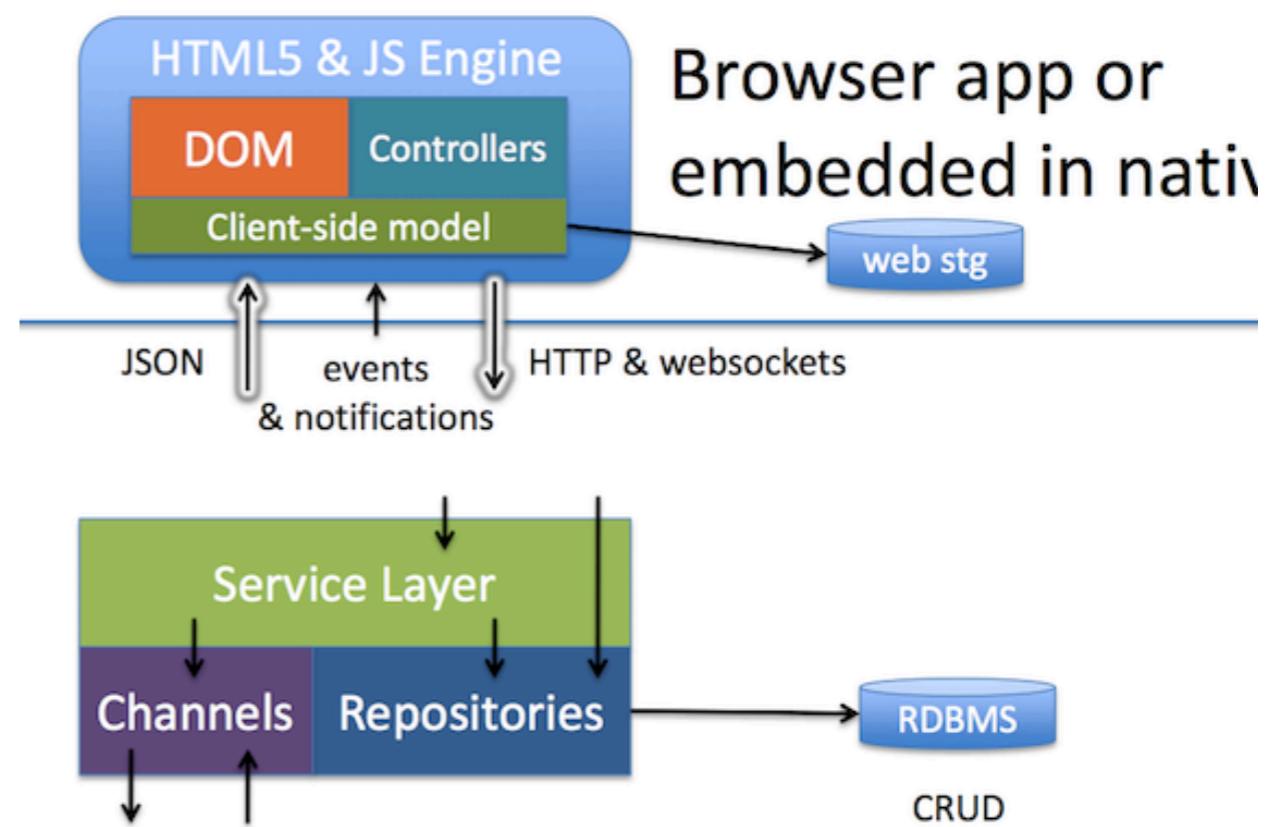


Gran parte del código puede pasar al cliente

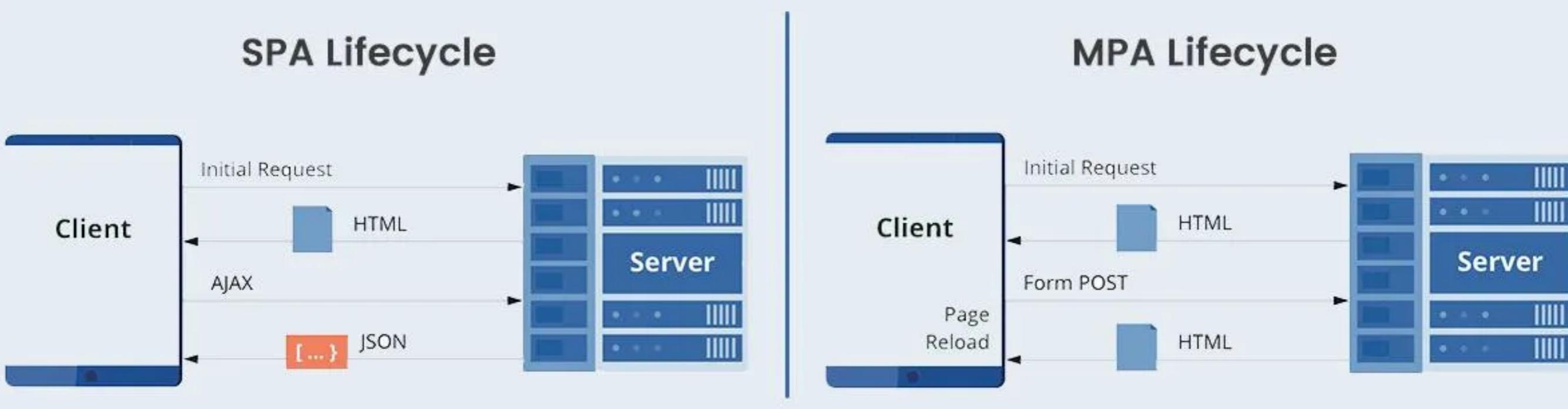
App “tradicional”



App basada en servicios



Y la app transformarse en SPA (Single Page App)



Alcanzaron su máxima popularidad en los 2010s

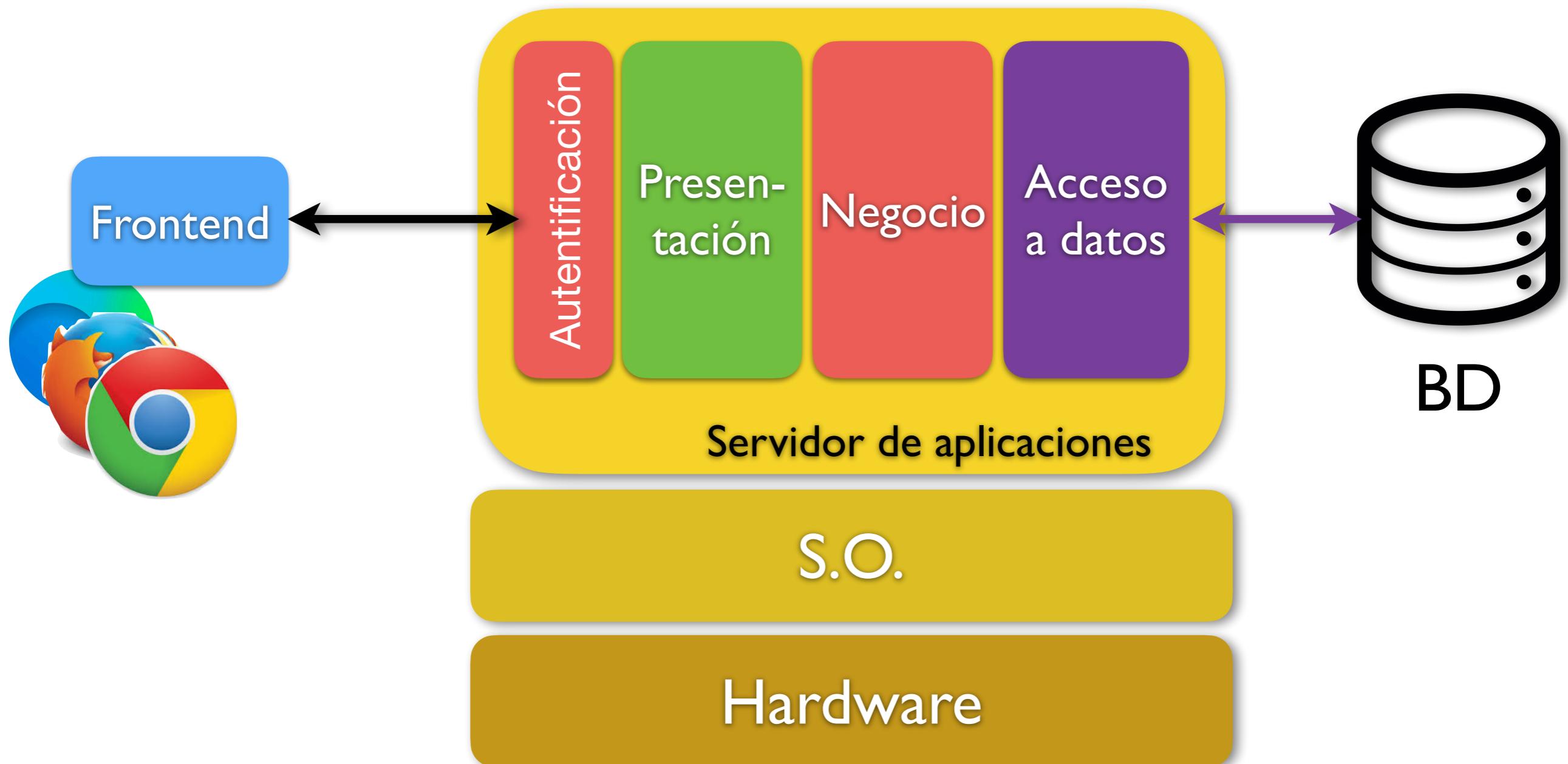
Tema 1: Introducción a las Aplicaciones web. Apps web
“en la nube”

1.2

Introducción a las Apps
“en la nube”
(IaaS vs PaaS vs BaaS vs FaaS)

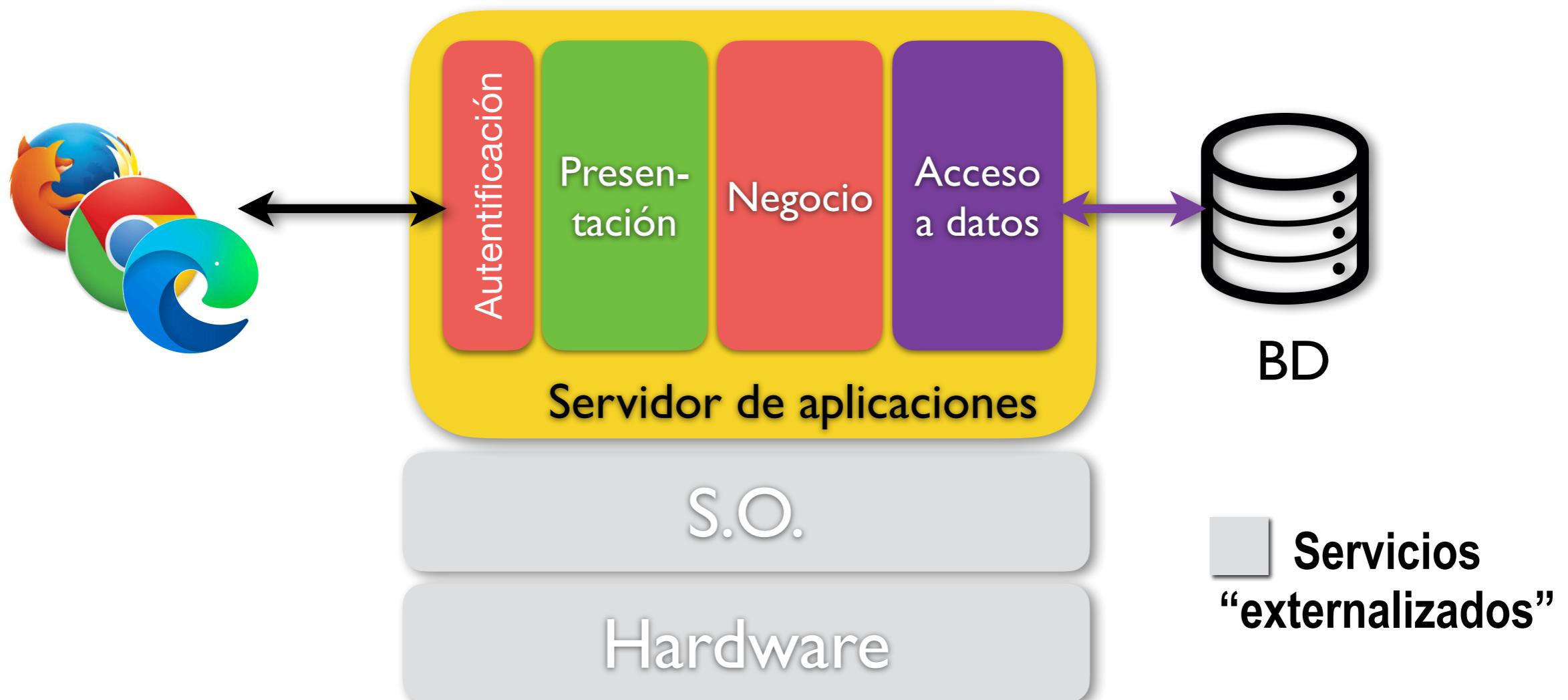
Aplicación clásica “sin nube”

Es nuestra responsabilidad no solo la **aplicación** sino también la **infraestructura**



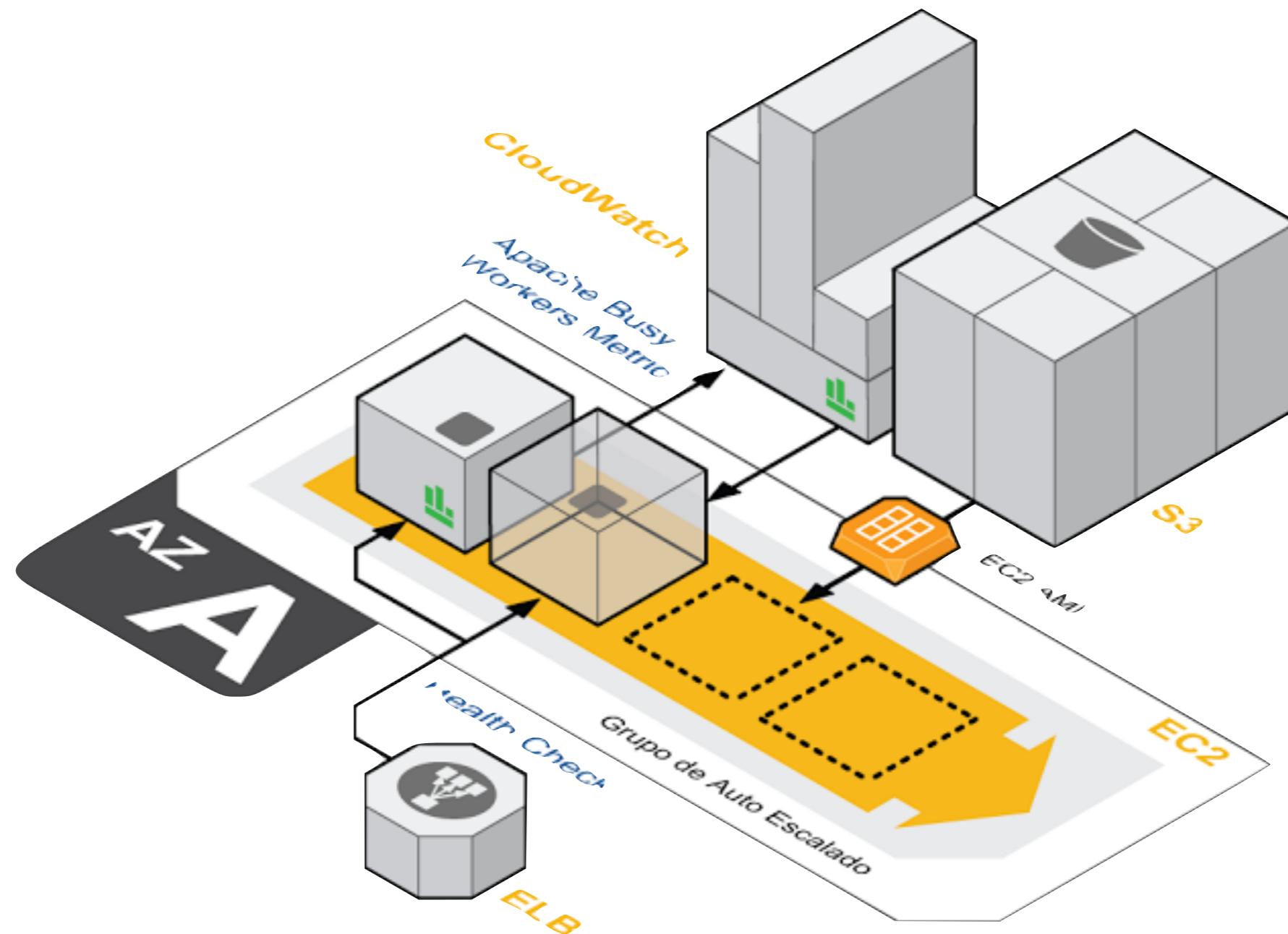
IaaS (Infrastructure as a Service)

- Se externaliza la **Infraestructura básica (Hardware + S.O.)**, aunque normalmente se usan máquinas virtuales. Seguimos teniendo que instalar/gestionar el software necesario (servidor y BD)
- Ejemplos: Amazon EC2, Azure, Rackspace,...



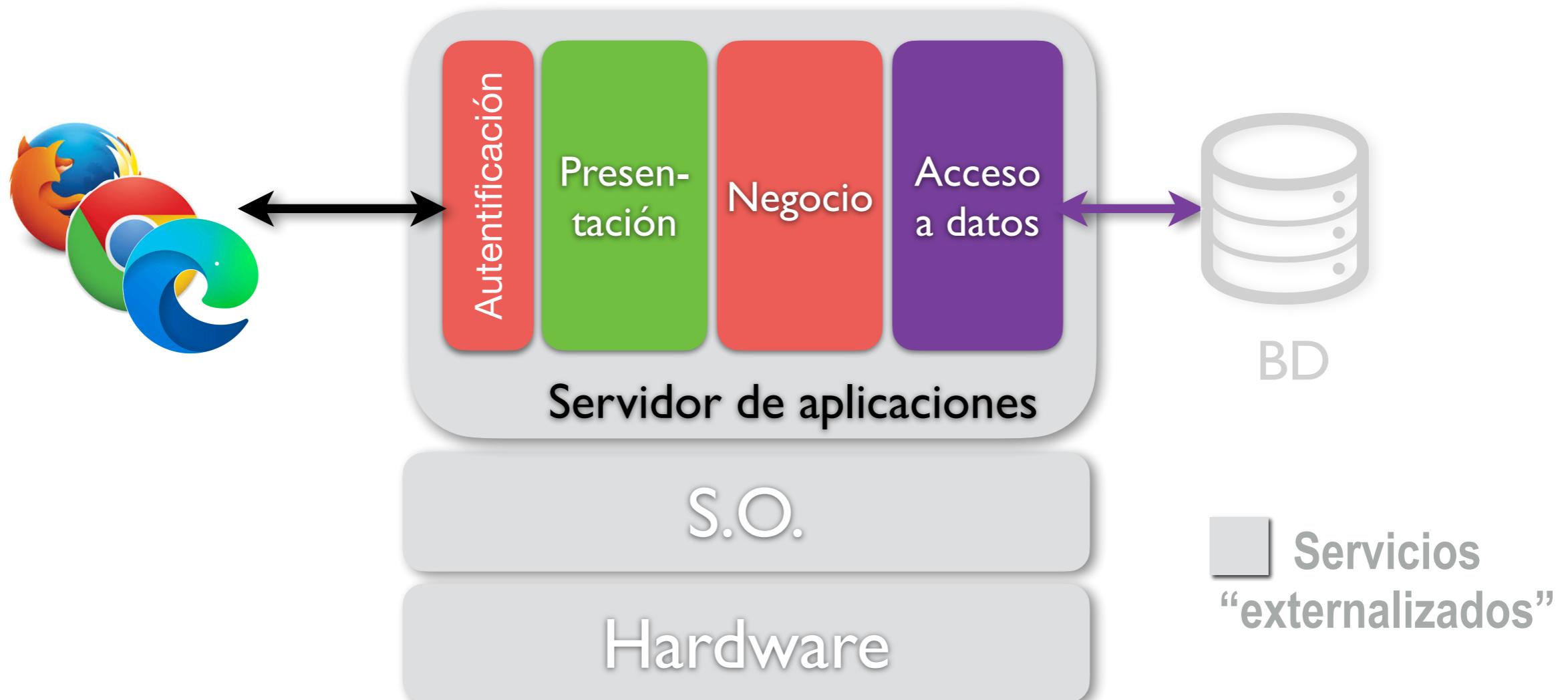
Servicios adicionales de IaaS

- Balanceadores de carga
- Autoescalado
- ...



PaaS (Platform as a Service)

- IaaS y además **soporte para ejecutar aplicaciones web** en distintos lenguajes: servidores web, bases de datos, ...
- Ejemplos: AWS Elastic Beanstalk, Azure, Heroku, Google App Engine, ...



Tema 1: Introducción a las Aplicaciones web. Apps web
“en la nube”

1.3

Backend as a Service

Aún con Paas el backend es complejo

- **Desarrollar** un *backend* y **desplegarlo** en una plataforma PaaS es más sencillo que gestionar nosotros la plataforma, pero **no es una tarea trivial**
- Muchos desarrolladores *frontend* **carecen de experiencia en backend**

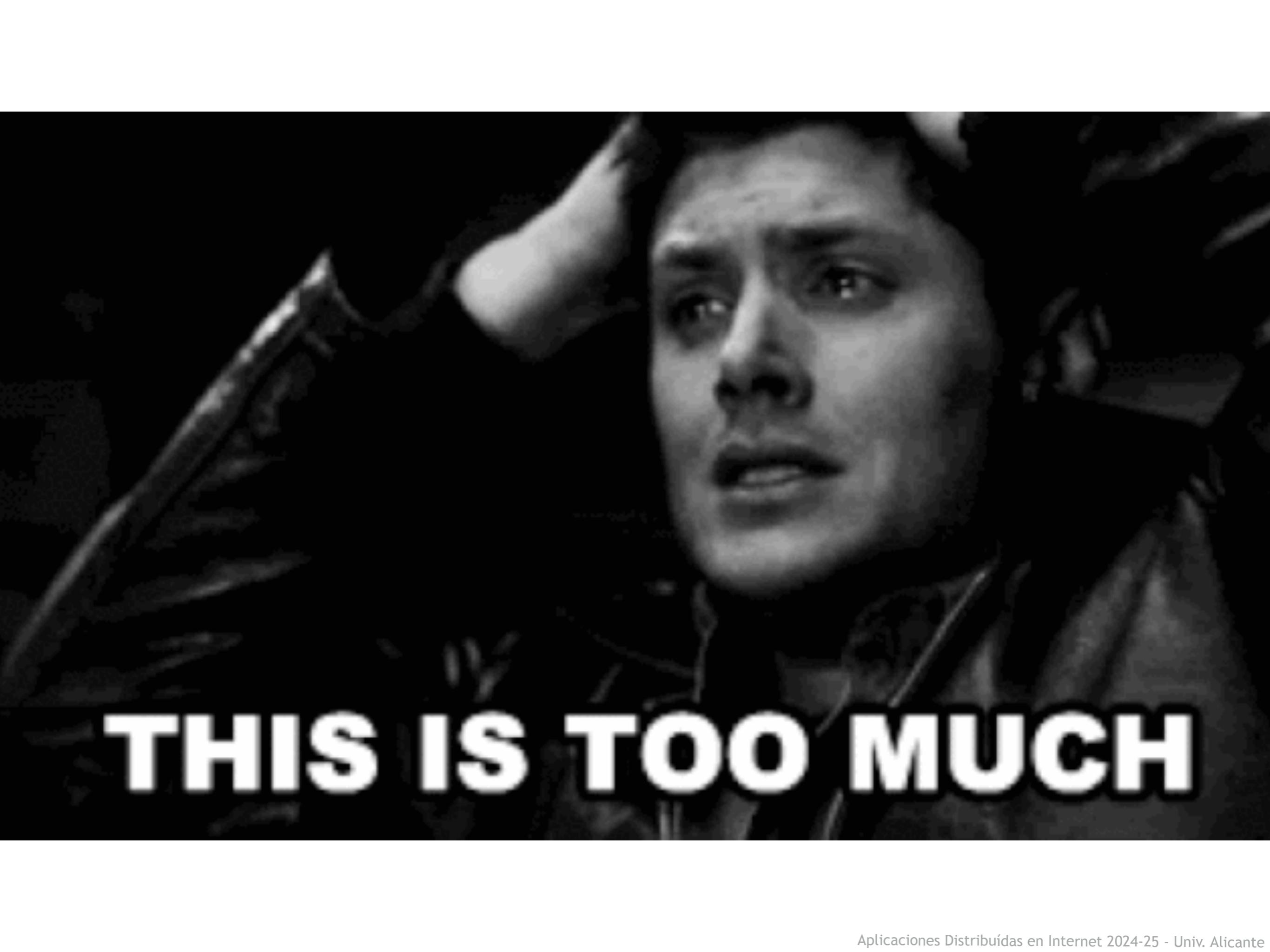


Ejemplo: gestión de usuarios

- Gestionar **la BD**
 - Crear y gestionar las tablas: “usuarios”, “roles”,...
 - Cifrar (*hash+salt*) los passwords, no pueden estar en claro en la BD
- Implementar los **casos de uso** asociados a la gestión de usuarios
 - Login, logout
 - CRUD de usuarios
- **Autentificación:**
 - Implementar algún “protocolo” como HTTP Basic o JWT
 - Uso de identidades de redes sociales (“Entra con tu cuenta de Facebook”) - OAuth
- **Pequeños detalles**
 - Enviar email para confirmar registro
 - Implementar reseteo de password por si a alguien se le olvida

Y eso sin contar con que todavía tenemos que implementar el **núcleo de nuestra aplicación**:

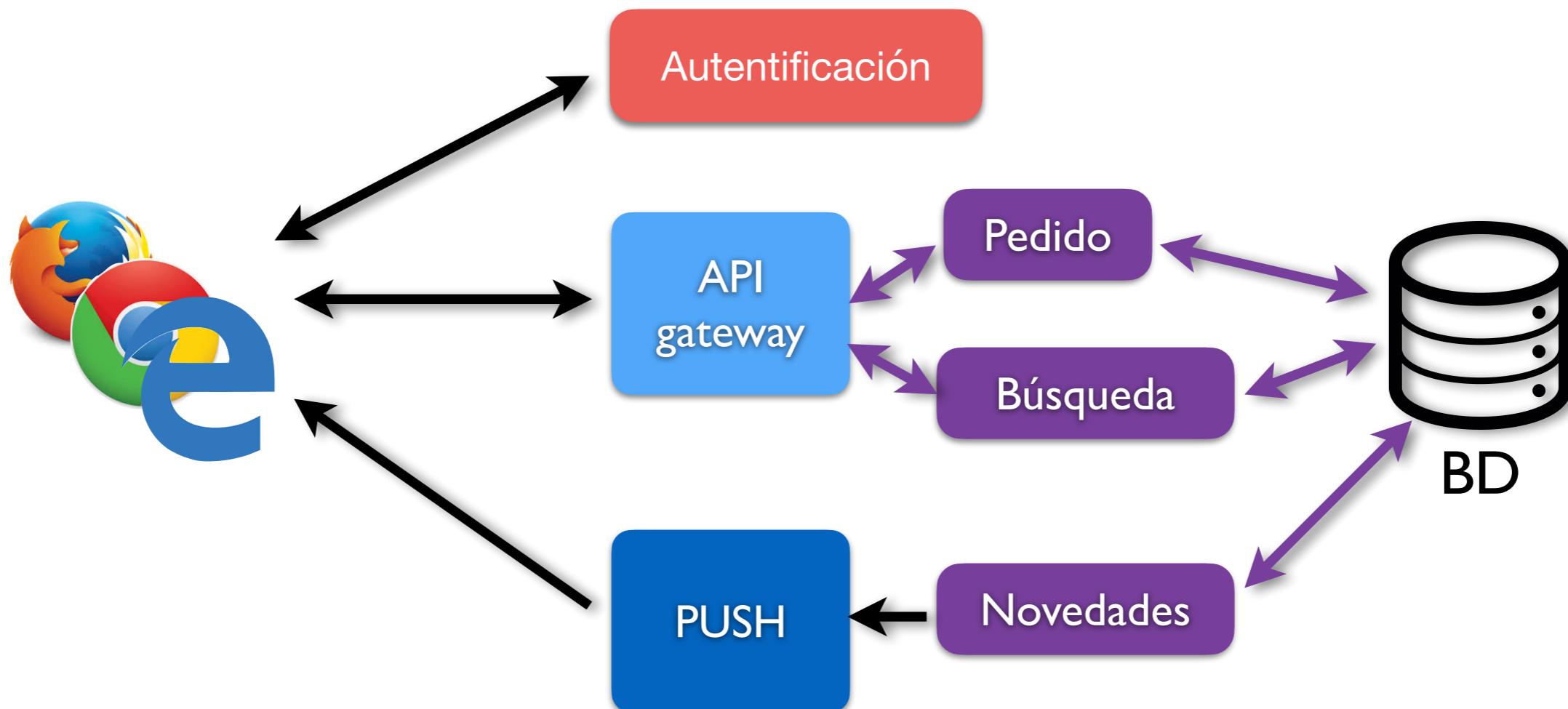
- Diseñar el **modelo del dominio**
- Implementar la **lógica de negocio**
- Implementar la **capa de acceso a datos**
- Crear y gestionar la **base de datos** (no el servidor, sino las tablas en sí)



THIS IS TOO MUCH

Serverless

- “Externalización” de los **servicios típicos de backend**
- No significa que no haya **servidor**, sino que es **transparente** para el desarrollador



Aunque no ponemos nada en gris TODO está “externalizado”. Nosotros solo escribimos el código de las partes en **■** y configuramos según necesidades el resto

Tipos de serverless

- **Backend as a Service**: externalización de los servicios típicos de *backend*
 - También llamado **Mobile Backend as a Service** para recalcar la idea de que son servicios muy apropiados para desarrolladores de apps móviles
 - Ejemplos: Firebase, Backendless, Back4App...
- **Functions as a Service**: externalización y modularización del soporte para la lógica de negocio, que acaba codificándose como un conjunto de funciones independientes
 - Ejemplos: Amazon Lambda, Azure Functions, ...
- Últimamente cuando se habla de **serverless** casi siempre se está hablando de **FaaS**



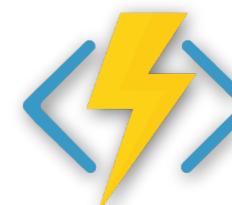
<https://www.back4app.com/>



Firebase
<https://firebase.google.com>



<https://aws.amazon.com/es/lambda>



<https://azure.microsoft.com/es-es/services/functions/>

Servicios típicos de BaaS

The screenshot shows the Supabase Docs website with a navigation bar at the top. The main content area is titled "Products" and lists six services: Database, Auth, Storage, AI & Vectors, Realtime, and Edge Functions. Below the products, there is a section for "Client Libraries" with icons for Javascript, Flutter, Python, C#, Swift, and Kotlin.

Products

- Database**
Supabase provides a full Postgres database for every project with Realtime functionality, database backups, extensions, and more.
- Auth**
Add and manage email and password, passwordless, OAuth, and mobile logins to your project through a suite of identity providers and APIs.
- Storage**
Store, organize, transform, and serve large files—fully integrated with your Postgres database with Row Level Security access policies.
- AI & Vectors**
Use Supabase to store and search embedding vectors.
- Realtime**
Listen to database changes, store and sync user states across clients, broadcast data to clients subscribed to a channel, and more.
- Edge Functions**
Globally distributed, server-side functions to execute your code closest to your users for the lowest latency.

Client Libraries

- Javascript
- Flutter
- Python
- C#
- Swift
- Kotlin

<https://supabase.com/docs>

Gracias a las plataformas **BaaS, desde el navegador** y usando **JavaScript** podemos registrar un nuevo usuario en el servidor de modo tan sencillo como

```
//En Firebase
var user = await firebase
    .auth()
    .createUserWithEmailAndPassword('adi@ua.es', '123456')
```

```
//En Parse
var user = new Parse.User();
user.set("username", "add");
user.set("password", "123456");
user.set("email", "adi@ua.es");
await user.signUp()
```

sin escribir **ni una línea de código en el servidor**

A photograph of Kermit the Frog, a green frog puppet, standing on a stage. He is wearing a light green suit jacket over a white shirt. Behind him is a red curtain. To his left is a black marquee sign with yellow lights around its perimeter. The sign displays the text "THE" in a small black font above "MUPPETS" in a large, bold, black font. The "M" in "MUPPETS" is stylized with a green gradient.

THE
MUPPETS

Plataformas BaaS especializadas

Se especializan en un tipo de servicio

- Comunicación en “**tiempo real**”/Servicios de notificación o **push**
- **Bases de datos** “as a service” (MongoDB Atlas, Cludbase, Cloudant, ...)
- **Autentificación** “as a service” (Auth0, SecureAuth, Okta,...)
- **Búsqueda** “as a service” (Algolia, Elastic,...)
- **Pagos** “as a service”.... (Stripe, FastSpring, Payoneer,...)

BaaS para “tiempo real”

- Ejemplo: <http://pubnub.com>: sistema pub/sub con baja latencia

PubNub®

```
var PUBNUB_demo = PUBNUB.init({
    publish_key: 'Your Publish Key Here',
    subscribe_key: 'Your Subscribe Key Here'
});
```

Inicialización

```
PUBNUB_demo.publish({
    channel: 'demoADI',
    message: {"texto": "hola PubNub"}
});
```

Envío

```
PUBNUB_demo.subscribe({
    channel: 'demoADI',
    message: function(m){console.log(m)}
});
```

Recepción

https://ottocol.github.io/ADI2021/teoria/arquitecturas/demos/demo_chat_pubnub.html

Autenticación como servicio

FREE JWT EBOOK!  Auth0

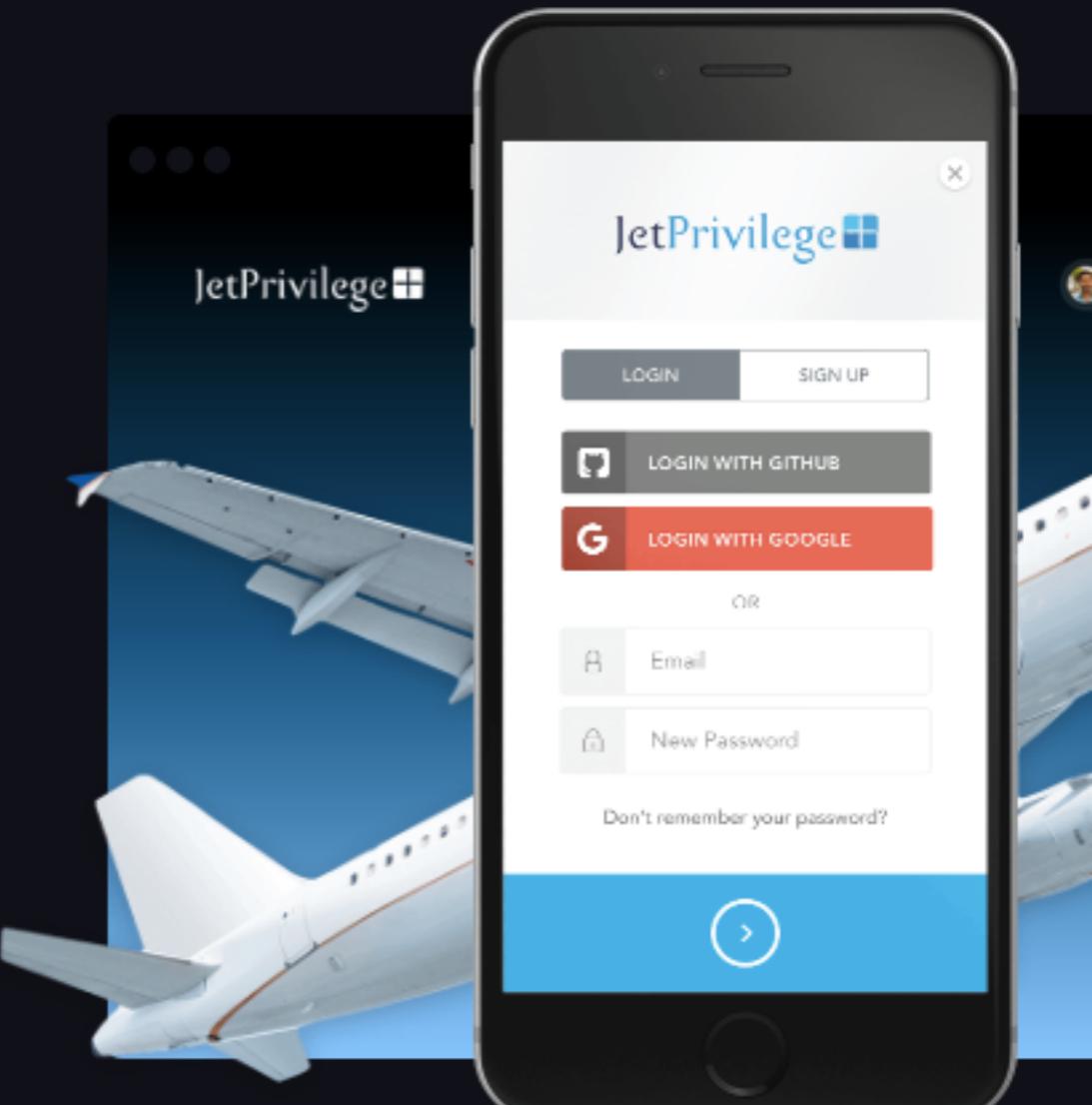
Products ▾ Why Auth0 ▾ Pricing Documentation More ▾ TALK TO SALES

Add authentication to your web and mobile apps in under 10 minutes

"Using Auth0, we'll offer users a richer experience through their social profile and personalized information."

Amol Date, JetPrivilege

TRYAUTH0 FOR FREE LEARN MORE



... A JS Apple Android Node.js C JS

Búsqueda como servicio



FEATURES PRICING ENTERPRISE ▾ DEVELOPER ▾



LOGIN OR

SIGN UP FREE

Build Unique Search Experiences

Hosted Search API that delivers instant and relevant results from the first keystroke

[SCHEDULE A DEMO](#)

[14-DAY FREE TRIAL ▾](#)

No credit card required.

X

GENRE

- Drama
- Comedy
- Action
- Thriller
- Romance
- Crime
- Family
- Music

8,714 MOVIES

Found in 15ms

ACTORS

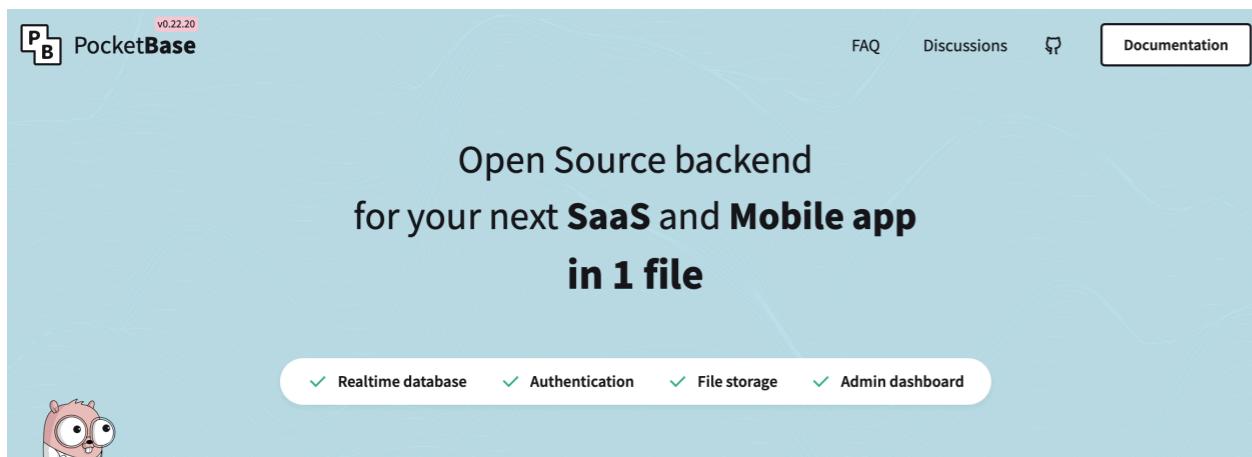
| Rank | Movie / Actor |
|------|----------------------|
| 1 | Fast and furious |
| 2 | Jennifer Lawrence |
| 3 | Intertstellar (with) |

Try to search for

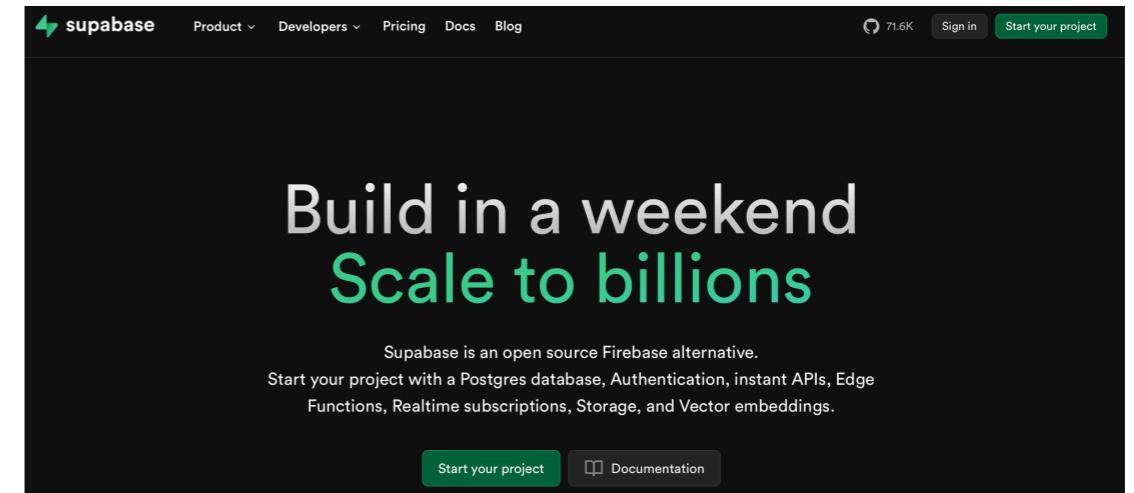
- 1 Fast and furious
- 2 Jennifer Lawrence
- 3 Intertstellar (with)

BaaS “instalable”

- Plataformas BaaS que podemos **instalar también en nuestra propia infraestructura**



<https://pocketbase.io>



<https://supabase.com>



The Complete Application Stack
Build applications faster with **object** and **file** storage,
user authentication, push notifications, dashboard and more out of the box.

<https://github.com/ParsePlatform/parse-server>

Tema 1: Introducción a las Aplicaciones web. Apps web
“en la nube”

1.4

Functions as a Service

Características

Beneficios de Lambda

No es necesario administrar servidores

Ejecute el código sin aprovisionar ni administrar la infraestructura. Simplemente escriba y cargue el código como un archivo .zip o una imagen de contenedor.

Escalamiento automático

Responda automáticamente a las solicitudes de ejecución de código a cualquier escala, desde una docena de eventos al día hasta cientos de miles por segundo.

Precio de pago por uso

Ahorre costos al pagar solamente por el tiempo de computación que utiliza, por milisegundo, en lugar de aprovisionar la infraestructura por adelantado para la capacidad máxima.

Optimización del desempeño

Optimice el tiempo de ejecución del código y el rendimiento con el tamaño adecuado de la memoria de las funciones. Responda a la alta demanda en milisegundos de dos dígitos con simultaneidad

<https://aws.amazon.com/es/lambda/>

Código de una func. en AWS

- El evento que ha disparado la función es el objeto **event**, y para devolver resultado llamamos a la función **callback**. El objeto **context** da información adicional sobre la propia función

```
console.log('Cargando funcion');

var mensajes = ['Hola', 'Qué tal,', 'Cómo te va,'];

exports.handler = function (event, context, callback) {
    var elegido = Math.floor(Math.random()*mensajes.length)
    var saludo = mensajes[elegido] + ' ' + event.nombre
    callback(null, saludo);
};
```

Código de una función en Firebase

```
// The Cloud Functions for Firebase SDK to create Cloud Functions and triggers.
const {logger} = require("firebase-functions");
const {onRequest} = require("firebase-functions/v2/https");
const {onDocumentCreated} = require("firebase-functions/v2/firestore");

// The Firebase Admin SDK to access Firestore.
const {initializeApp} = require("firebase-admin/app");
const {getFirestore} = require("firebase-admin/firestore");

initializeApp();

// Take the text parameter passed to this HTTP endpoint and insert it into
// Firestore under the path /messages/:documentId/original
exports.addmessage = onRequest(async (req, res) => {
  // Grab the text parameter.
  const original = req.query.text;
  // Push the new message into Firestore using the Firebase Admin SDK.
  const writeResult = await getFirestore()
    .collection("messages")
    .add({original: original});
  // Send back a message that we've successfully written the message
  res.json({result: `Message with ID: ${writeResult.id} added.`});
});
```

En la terminal: `firebase deploy --only functions`

Function URL (addMessage): https://us-central1-MY_PROJECT.cloudfunctions.net/addMessage

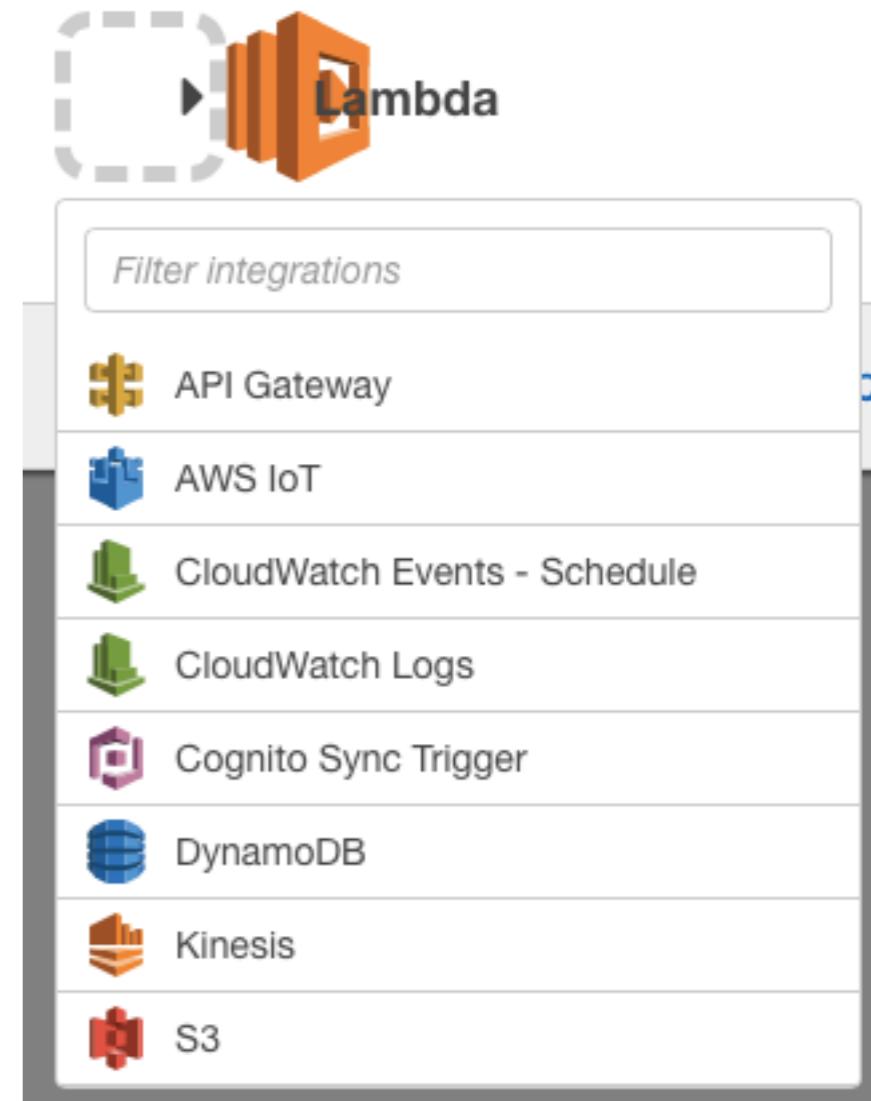
Firebase asocia cada función con una URL bajo el proyecto

Eventos para disparar la función

Las plataformas suelen aceptar peticiones HTTP y también conectan con los servicios propios.

Por ejemplo en Amazon:

- Petición HTTP (API Gateway)
- Eventos sobre una BD DynamoDB (p.ej. nuevo registro)
- Eventos sobre un *bucket* de S3 (Simple Storage Service). Por ejemplo, subir un archivo
- Eventos sobre un dispositivo IoT (Internet of Things)
- ...



El despliegue es complejo

- Por desgracia actualmente el **despliegue** y **configuración** de FaaS **no es tan sencillo** como debería ser
- Además el proceso es **distinto** según la **plataforma**
- Hay algunos *frameworks* que intentan solucionar estos problemas
 - **Serverless framework**: soporta AWS, Google Cloud, Azure ...
 - **ClaudiaJS**: simplifica el despliegue pero solo para AWS <https://claudiajs.com/>



<https://serverless.com/>



<https://claudiajs.com/>

Limitaciones

- **Recursos** limitados
 - Tiempo de cómputo máximo 15 minutos. Esto impide por ejemplo, usar lambda para trabajos en *background* o para algunos trabajos en *batch*
 - Memoria máxima (por defecto 512Mb, aunque ampliable hasta unos 10Gb)
- **Latencia** en la ejecución
 - Una lambda escrita en Java necesita que arranque la JVM, lo que es costoso. Se deja “arrancada” la JVM para evitarlo pero si pasa mucho tiempo entre invocación e invocación, libera la memoria. Esto no es tan problemático con Javascript/Python (blog post: [Analyzing cold start latency of AWS Lambda](#))
- **Sin estado:** no podemos asumir que se guarde nada entre ejecución y ejecución. Si queremos guardar algo permanente debemos hacerlo por ejemplo en un servicio de BD

P: ¿Por qué las funciones de AWS Lambda no deben tener estado?

Mantener las funciones sin estado permite a AWS Lambda lanzar rápidamente tantas copias de la función como resulten necesarias para escalar hasta la tasa de eventos entrantes. A pesar de que el modelo de programación de AWS Lambda no tiene estado, el código puede obtener acceso a datos con estado al llamar a otros servicios web, como Amazon S3 o Amazon DynamoDB.

Autoescalado

La plataforma se encarga de **crear automáticamente las copias necesarias** para atender peticiones en paralelo (con un máximo actualmente de 1000)

- **Event sources that aren't stream-based** – If you create a Lambda function to process events from event sources that aren't stream-based (for example, Amazon S3 or API Gateway), each published event is a unit of work. Therefore, the number of events (or requests) these event sources publish influences the concurrency.

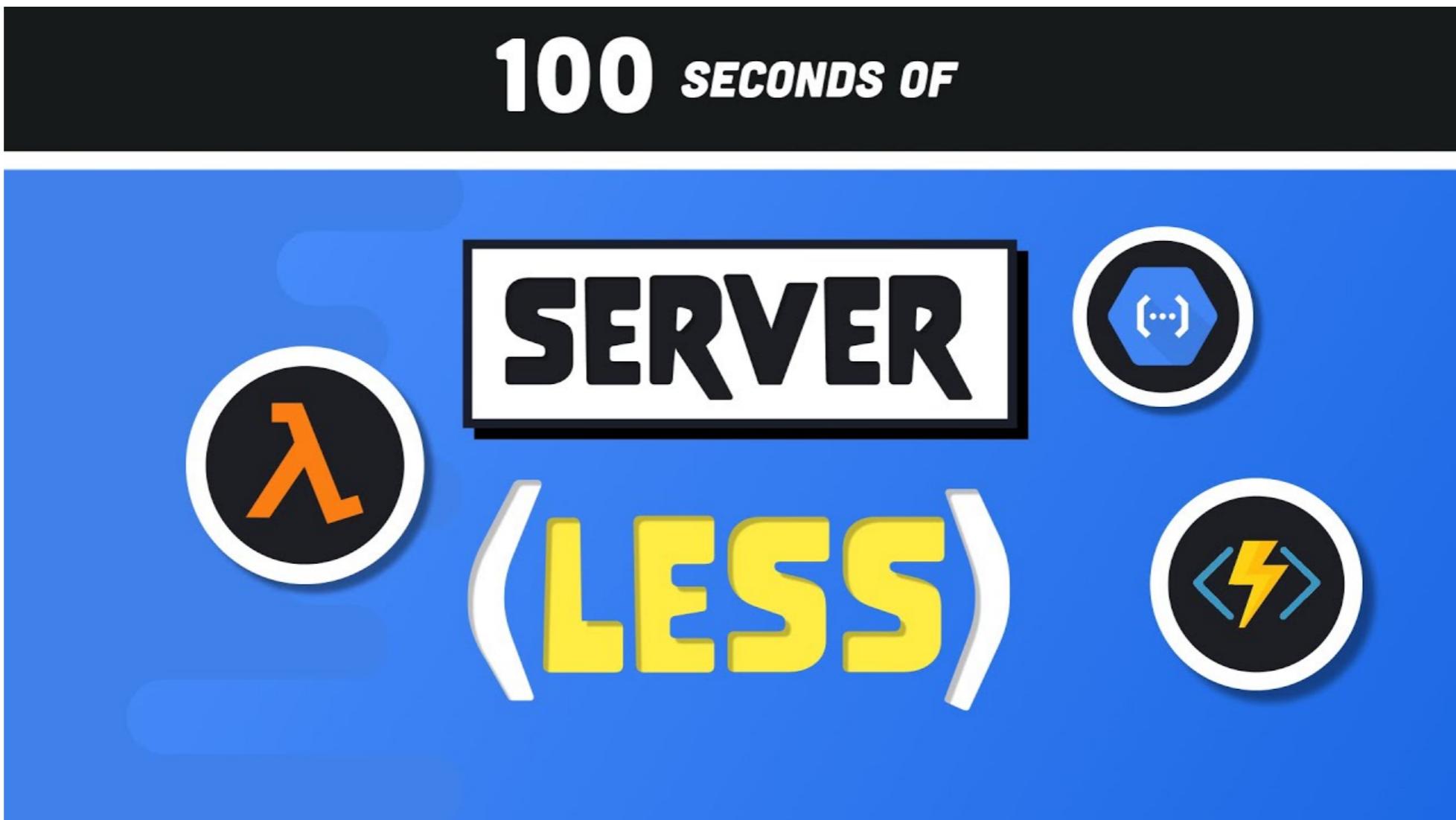
You can use the following formula to estimate your concurrent Lambda function invocations:

```
events (or requests) per second * function duration
```

For example, consider a Lambda function that processes Amazon S3 events. Suppose that the Lambda function takes on average three seconds and Amazon S3 publishes 10 events per second. Then, you will have 30 concurrent executions of your Lambda function.

<http://docs.aws.amazon.com/lambda/latest/dg/concurrent-executions.html>

Tutorial para una app completa



https://www.youtube.com/watch?v=W_VV2Fx32_Y

Tema 1: Introducción a las Aplicaciones web. Apps web
“en la nube”

1.5 Conclusiones

Ventajas de la nube

- **Simplificación** de la gestión de operaciones
- Reducción de **costes** en mantenimiento de infraestructura
- Escalabilidad

DHH ✅ @dhh · 14 sept.

"[ziglang.org](#) used to be hosted on Amazon S3 + CloudFront, but as it became more expensive to run due to increased traffic, we have migrated to a one-computer solution for the website and tarball hosting." Hetzner delivers again! [ziglang.org/news/migrate-t...](#)

23 101 1 mil 186 mil

No siempre Serverless equivale a reducción de costes

DHH ✅ @dhh · 12 sept.

Serverless makes no sense when you regularly need a whole server. (And that bill could be compressed a lot further outside of GCP!).

Théo Champion 🌟 ✅ @deletethistheo · 11 sept.

Just saved \$10k/month on our cloud bill by moving our main media processing service from serverless (Cloud Run) to a single mega vps (64 cores, 128Gb RAM)

I'm starting to understand @dhh and the move out of serverless

```
root@leviathan:~# neofetch
$ $gg.          root@leviathan
$ $$$$$$P.      -----
" "Y$$. ".     '$$$.
` $$b:          '$$.
' .           '$$.
' ,           '$$.
- ,d$$'        '$$.
- ,d$P'        '$$P'
$ $$P"          '$$P'

OS: Debian GNU/Linux 11 (bullseye)
Host: Google Compute Engine
Kernel: 6.1.100+
Uptime: 11 hours, 45 mins
Packages: 282 (dpkg)
Shell: bash 5.1.4
Terminal: systemd-nspawn
CPU: AMD EPYC 7B13 (128) @ 2.4GHz
Memory: 48149MiB / 128886MiB
```

40 56 749 109 mil

Problemas

- **Dependencia** de la plataforma:
 - Prácticamente nula portabilidad
 - ...¿qué pasa si desaparece?. Ya ha pasado alguna vez
- Tecnologías **inmaduras**:
 - Se necesitan **mejores herramientas** para despliegue e integración
 - Todavía no hay un conjunto de **buenas prácticas** aceptadas a la hora de estructurar la aplicación, sobre todo en FaaS

Algunas referencias

- <http://martinfowler.com/articles/serverless.html>
 - Artículo de un colaborador de Fowler que explica bastante bien y de manera neutral lo que significa *serverless*, sus implicaciones en cuanto a la arquitectura de las aplicaciones y sus ventajas e inconvenientes
- <https://github.com/anaibol/awesome-serverless>
 - Extensa lista de todo tipo de recursos sobre *serverless*, sobre todo de plataformas que ofrecen estos servicios, pero también de frameworks de desarrollo, libros, artículos introductorios...
- Sitio web/ebook: <https://serverless-stack.com/>

