

Aplicaciones Distribuidas en Internet

Tema 5:
**Aplicaciones web “en la
nube”**

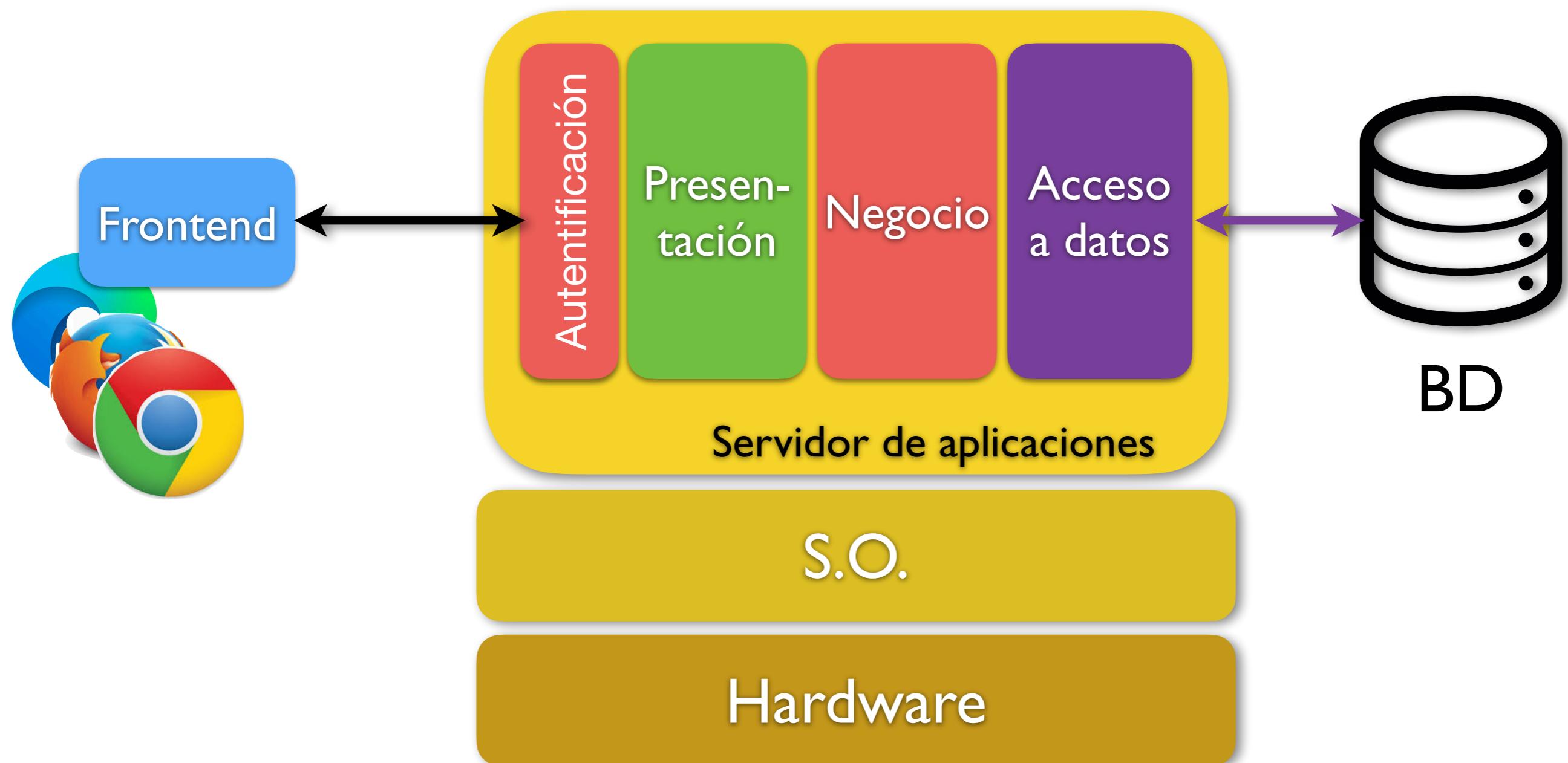
Tema 6: Aplicaciones web en “la nube”

5.1

**Introducción:
IaaS vs PaaS vs BaaS
vs FaaS vs ...**

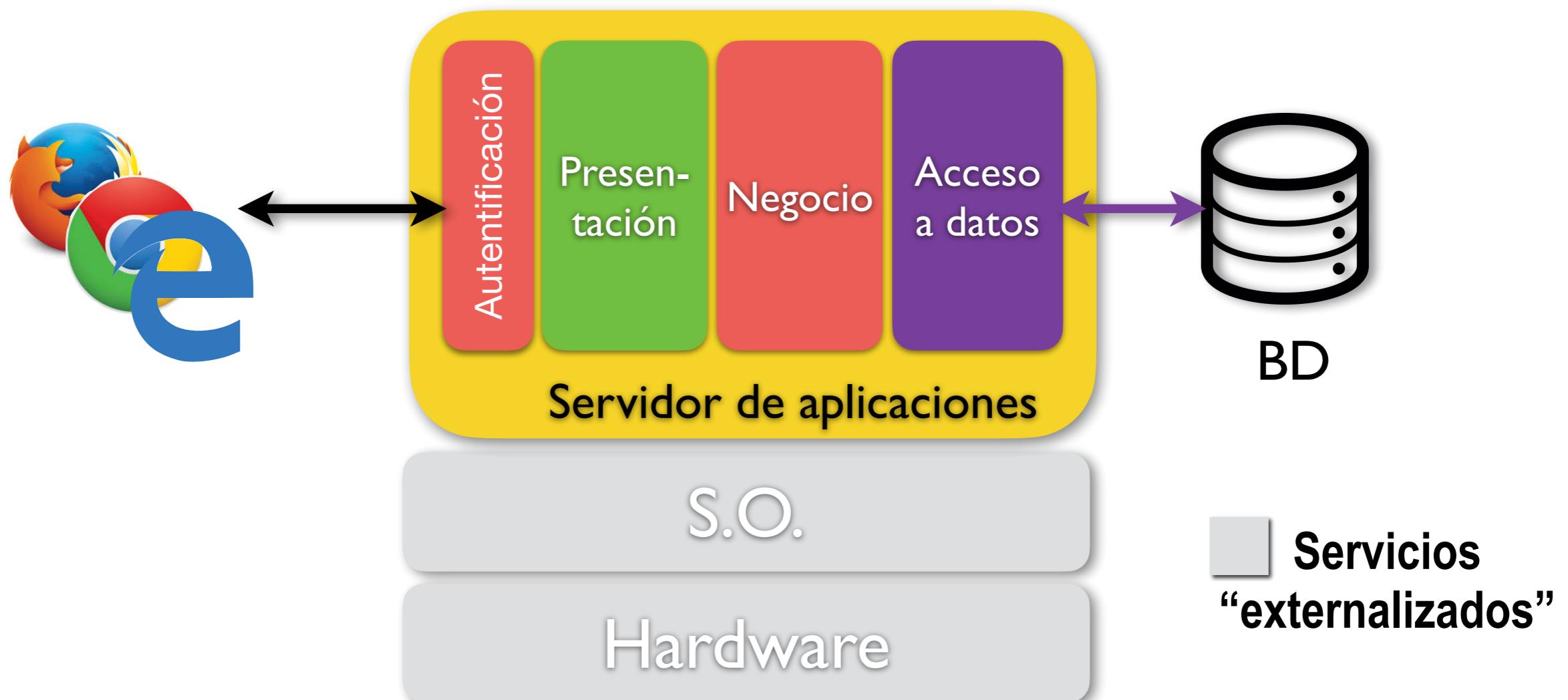
Aplicación clásica sin “nube”

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



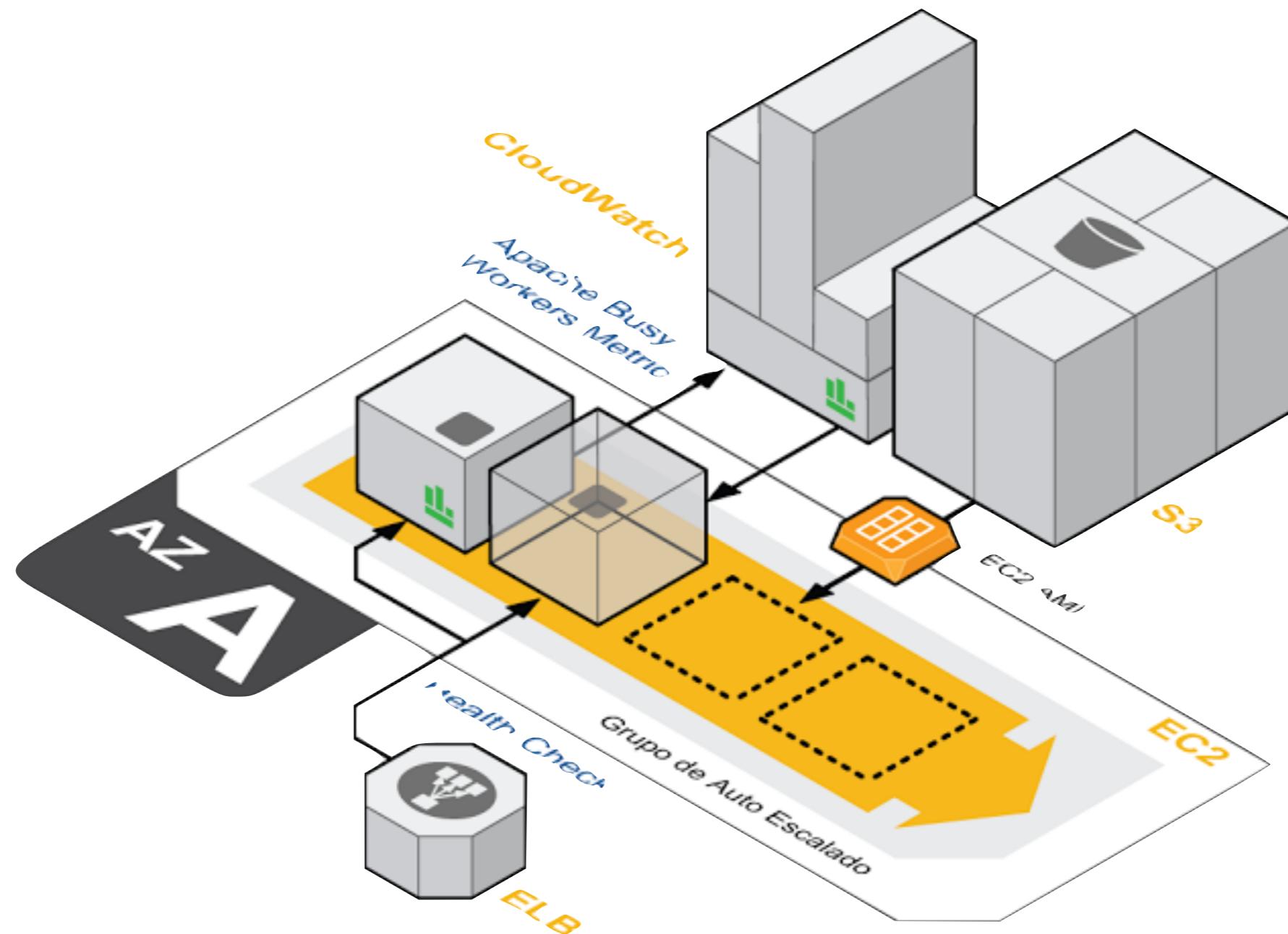
IaaS (Infrastructure as a Service)

- 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,...



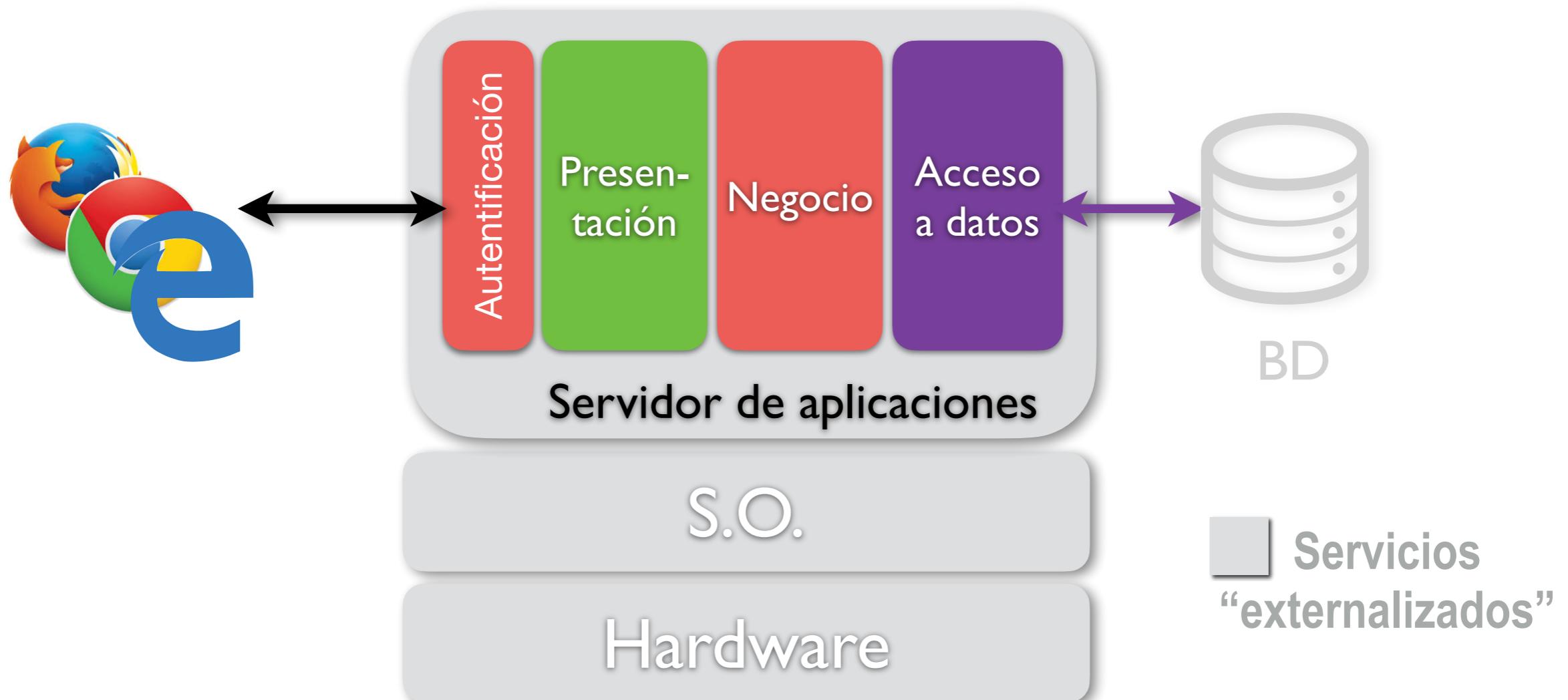
Servicios adicionales de IaaS

- 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 6: Aplicaciones web en “la nube”

5.2 (Mobile) Backend as a Service

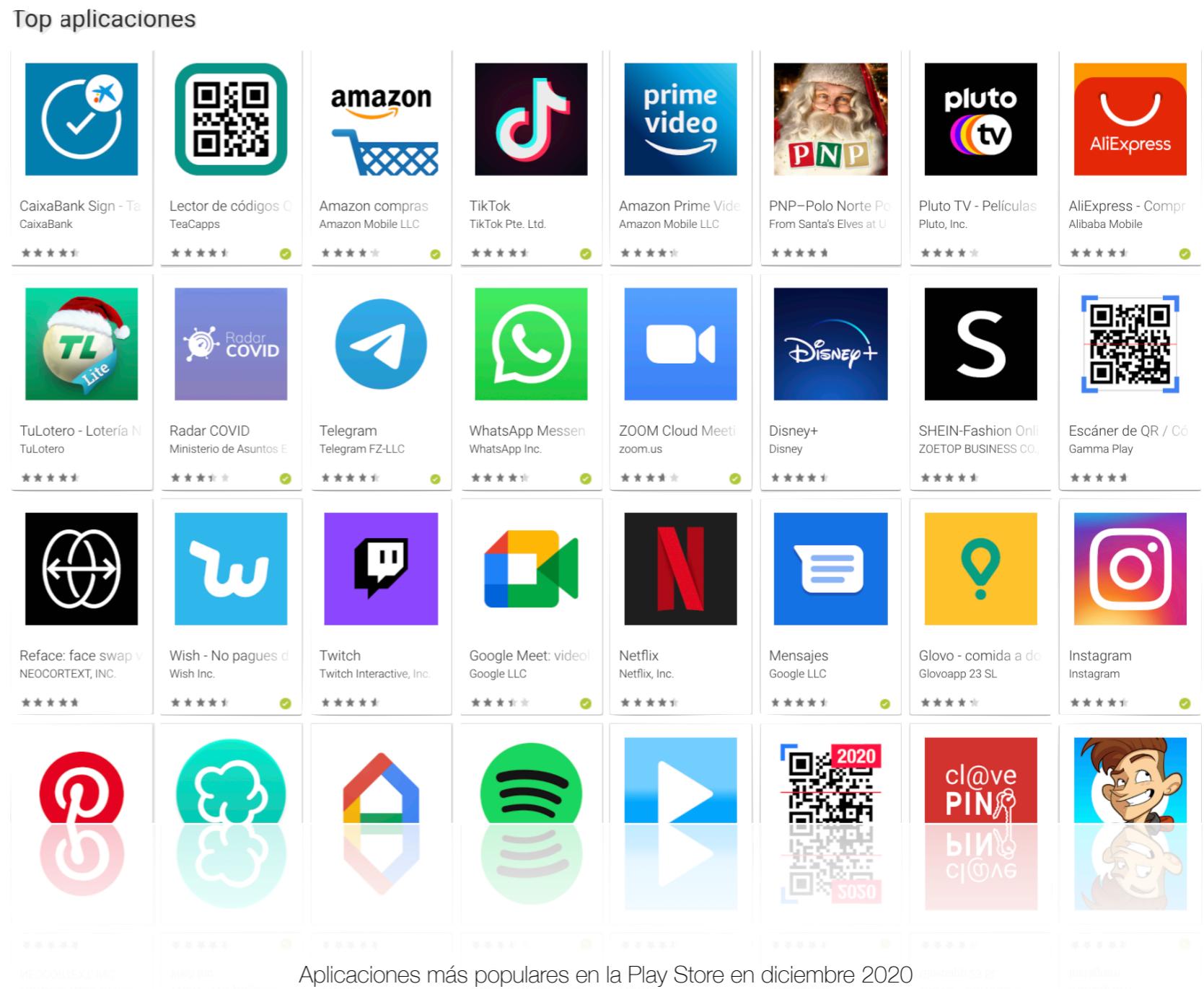
Aún con Paas el backend es complejo

- **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**



Backend para apps móviles

- No solo las apps web necesitan *backend*
- Los desarrolladores de apps móviles suelen estar todavía más “perdidos” en *backend* que los de *frontend* web



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**:

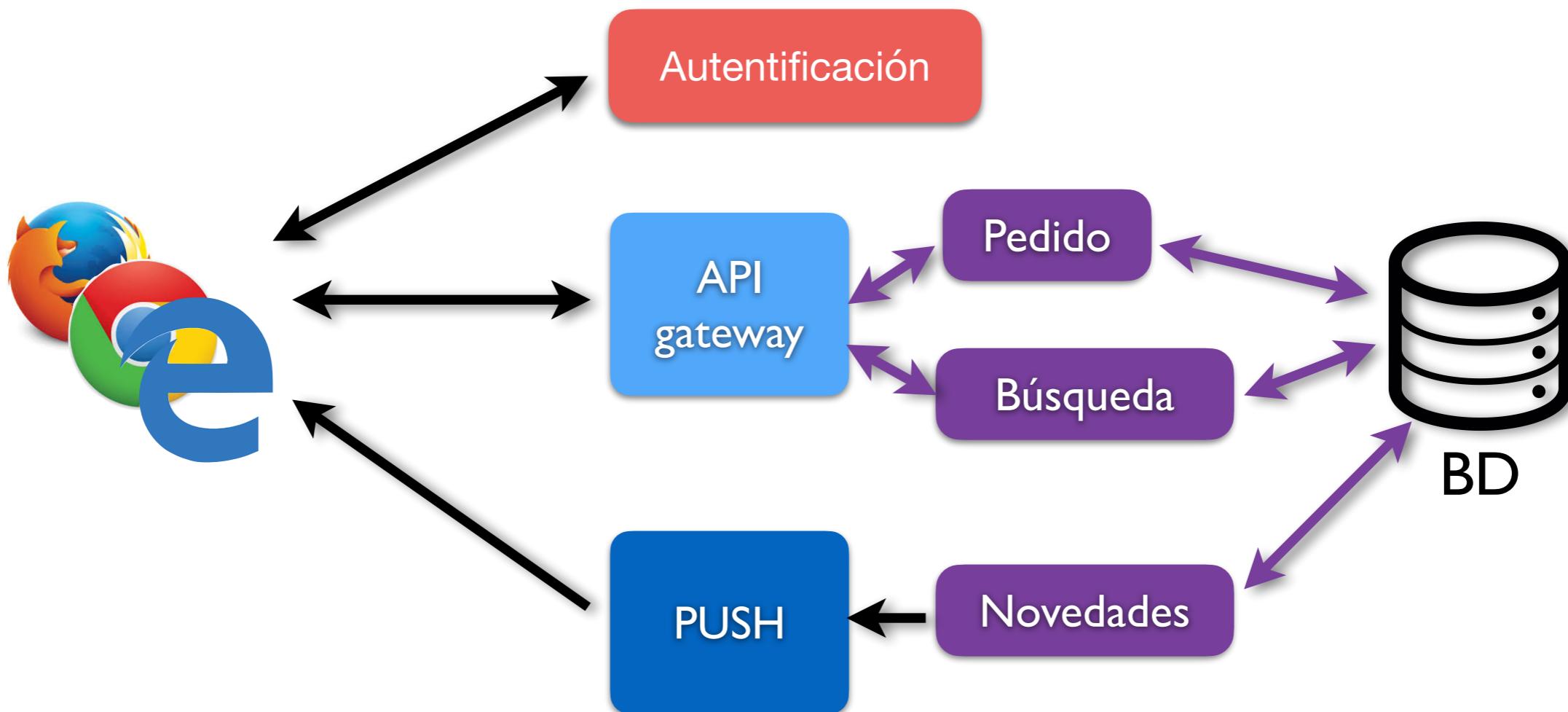
- Diseñar el **modelo del dominio**
- Implementar la **lógica de negocio**
- Implementar la **capa de acceso a datos**
- Crear y gestionar la **base de datos** (no el servidor, sino las tablas en sí)



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 **Novedades** 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**



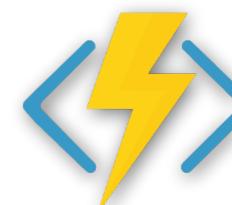
<https://www.back4app.com/>



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



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



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

Servicios típicos de BaaS

Info de Firebase: <https://firebase.google.com/products-build>



Authentication

Add an end-to-end, customizable identity solution to your app in under 10 lines of code.

[Más información](#)

Autentificación y gestión de usuarios



Cloud Messaging

Get infrastructure to reliably send and receive push messages between your server and devices, across platforms at no cost.

[Más información](#)

Notificaciones push



Cloud Functions

Write and run app logic server-side without needing to set up your own server.

[Más información](#)

Lógica de negocio en el servidor



Realtime Database

Build serverless apps by storing and syncing JSON data between your users in near-realtime, on or offline, with strong user-based security.

[Más información](#)

Persistencia



Cloud Storage

Store and serve app content with ease as your app grows from prototype to production.

[Más información](#)

Hosting de la web y almacenamiento de archivos



Google Analytics

Monitor how your latest release is being adopted by users.

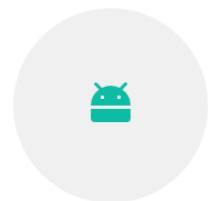
[Más información](#)

Analíticas, Dashboard, paneles de control...

SDKs multiplataforma

Firebase por plataforma

Firebase te da las herramientas para programar apps de alta calidad, aumentar tu base de usuarios y ganar más dinero. Nos encargamos de los aspectos básicos para que puedas monetizar tu negocio y concentrarte en tus usuarios.



Comienza a usar Android



Referencia de la API

Codelabs



Comienza a usar iOS



Referencia de la API

Codelabs



Comienza a usar Web



Referencia de la API

Codelabs



Comienza a usar C++



Referencia de API



Comienza a usar Unity



Referencia de API

Comienza a usar Admin



Referencia de API

<https://firebase.google.com/docs?hl=es-419>

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

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

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

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

A photograph of Kermit the Frog, a green frog puppet, sitting 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 sign with yellow lights around its perimeter. The sign has the words "THE" in small black letters above "MUPPETS" in large black letters. The letter "M" in "MUPPETS" is stylized with a green gradient. The stage floor is dark.

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

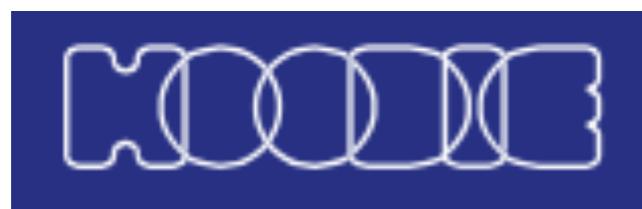
```
PUBNUB_demo.subscribe({
    channel: 'demoADI',
    message: function(m){console.log(m)}
});
```

Recepción

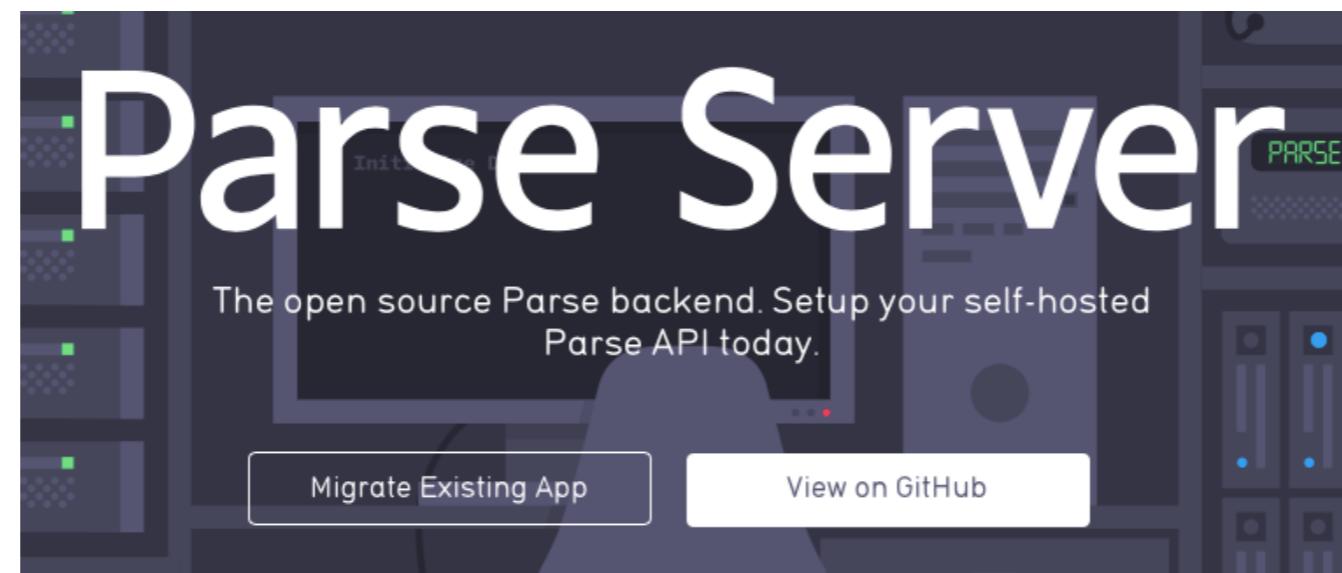
https://ottocol.github.io/ADI2021/teoria/arquitecturas/demos/demo_chat_pubnub.html

BaaS “instalable”

- Plataformas BaaS que podemos **instalar en nuestra propia infraestructura** en lugar de “en la nube”



Hoodie



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

Tema 6: Aplicaciones web en “la nube”

5.3

Functions as a Service (FaaS)

Características de FaaS

AWS Lambda le permite ejecutar código sin aprovisionar ni administrar servidores. Solo pagará por el tiempo de cómputo que consuma – no se cobra nada cuando el código no se está ejecutando. Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio back-end – y todo sin administrar nada. Usted solo tiene que cargar el código. Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad. Puede configurar el código para que se active automáticamente desde otros servicios de AWS o puede llamarlo directamente desde cualquier aplicación web o móvil

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

Características de FaaS

AWS Lambda le permite ejecutar código **sin aprovisionar ni administrar servidores**. Solo pagará por el tiempo de cómputo que consuma – no se cobra nada cuando el código no se está ejecutando. Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio back-end – y todo sin administrar nada. Usted solo tiene que cargar el código. Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad. Puede configurar el código para que se active automáticamente desde otros servicios de AWS o puede llamarlo directamente desde cualquier aplicación web o móvil

- Serverless

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

Características de FaaS

AWS Lambda le permite ejecutar código **sin aprovisionar ni administrar servidores**. Solo pagará por el tiempo de cómputo que consuma – **no se cobra nada cuando el código no se está ejecutando**. Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio back-end – y todo sin administrar nada. Usted solo tiene que cargar el código. Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad. Puede configurar el código para que se active automáticamente desde otros servicios de AWS o puede llamarlo directamente desde cualquier aplicación web o móvil

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

- Serverless
- No se cobra por tiempo que el servidor está *online*

Características de FaaS

AWS Lambda le permite ejecutar código sin aprovisionar ni administrar servidores. Solo pagará por el tiempo de cómputo que consuma – no se cobra nada cuando el código no se está ejecutando. Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio back-end – y todo sin administrar nada. Usted solo tiene que cargar el código. Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad. Puede configurar el código para que se active automáticamente desde otros servicios de AWS o puede llamarlo directamente desde cualquier aplicación web o móvil

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

- Serverless
- No se cobra por tiempo que el servidor está *online*
- Escalado horizontal automático. Se ejecutará en paralelo el número de copias necesarias

Características de FaaS

AWS Lambda le permite ejecutar código sin aprovisionar ni administrar servidores. Solo pagará por el tiempo de cómputo que consuma – no se cobra nada cuando el código no se está ejecutando. Con Lambda, puede ejecutar código para casi cualquier tipo de aplicación o servicio back-end – y todo sin administrar nada. Usted solo tiene que cargar el código. Lambda se encargará de todo lo necesario para ejecutar y escalar el código con alta disponibilidad. Puede configurar el código para que se active automáticamente desde otros servicios de AWS o puede llamarlo directamente desde cualquier aplicación web o móvil

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

- Serverless
- No se cobra por tiempo que el servidor está *online*
- Escalado horizontal automático. Se ejecutará en paralelo el número de copias necesarias
- Las funciones se activan con eventos disparados por otros servicios (p.ej. una cola de mensajes, una BD, ...) o bien por una petición HTTP

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

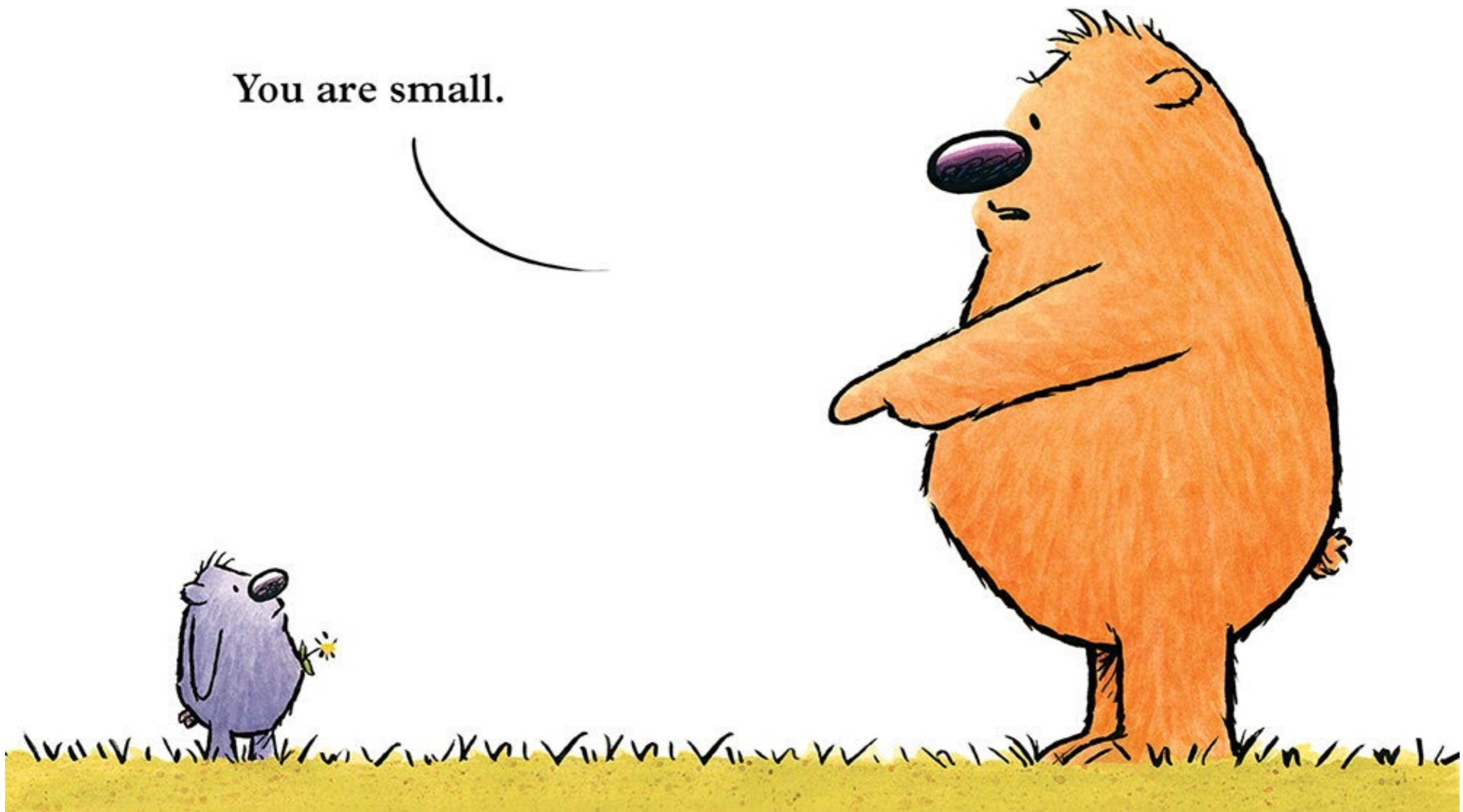
```
console.log('Cargando funcion');

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);
};
```

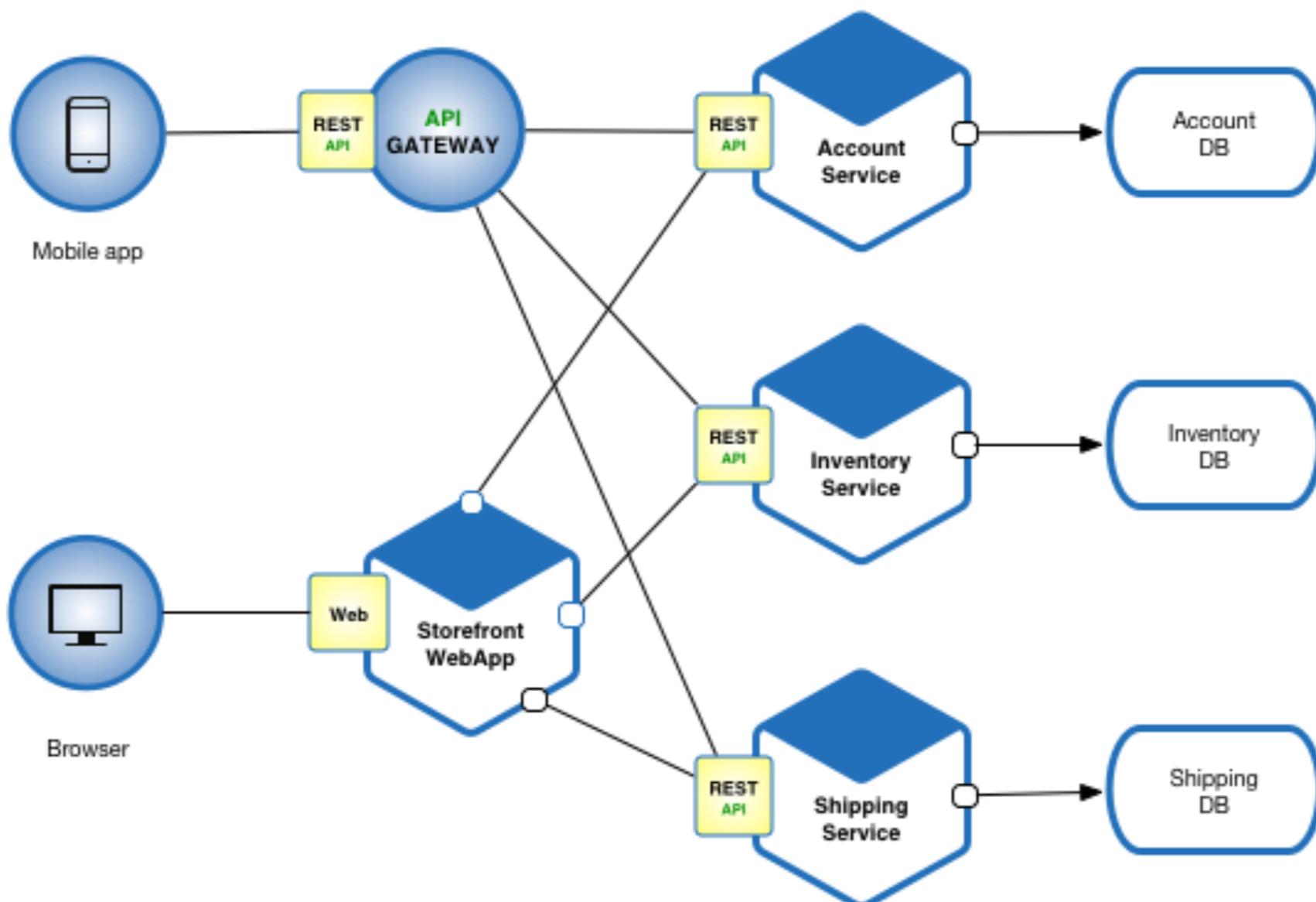
Pero esto es solo una función, y una aplicación en el servidor debe tener **mucho** más código ¿no?

You are small.



Microservicios

- Arquitectura en la que en lugar de un código monolítico en el servidor tenemos muchos servicios “pequeños”, independientes entre sí y cada uno con su propia BD

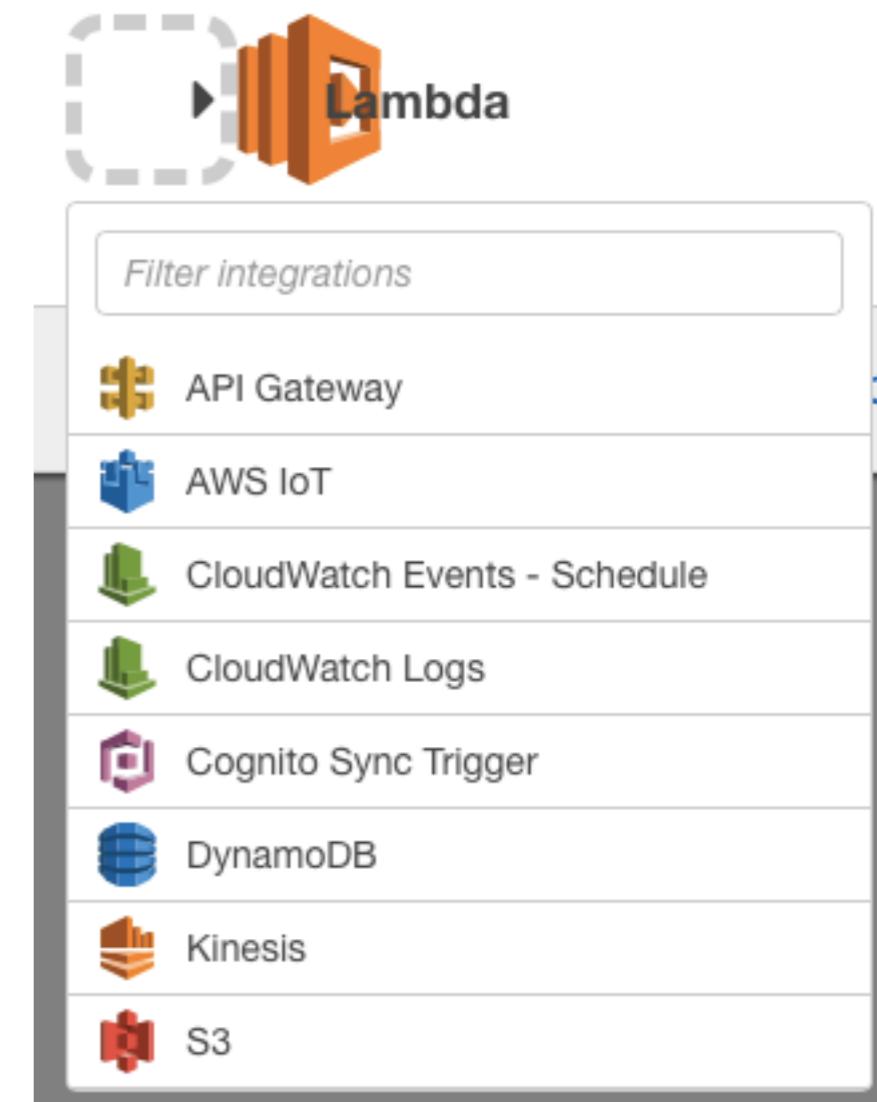


Eventos para disparar la función

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 128Mb, 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 100)

- **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:

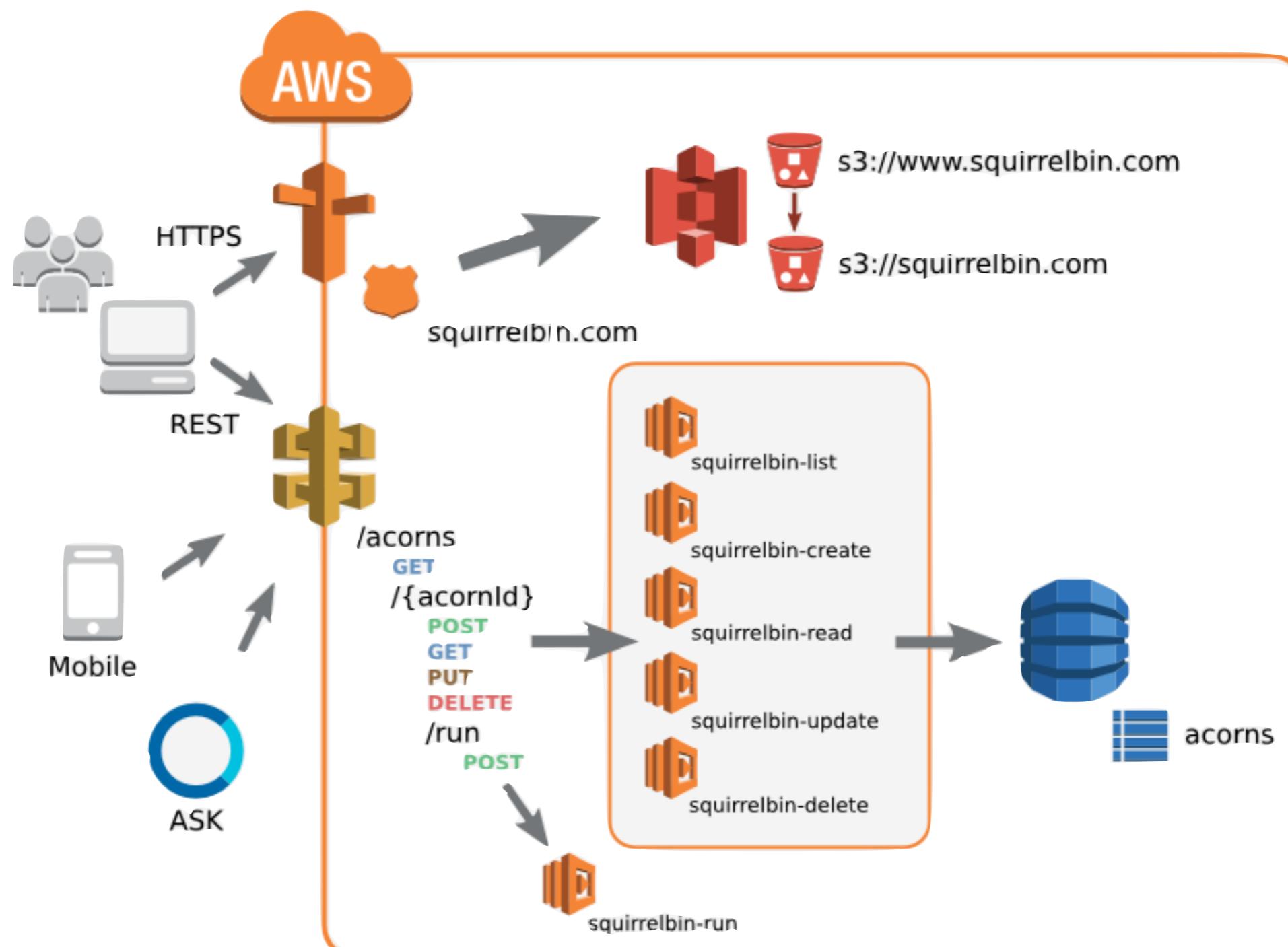
```
events (or requests) per second * function duration
```

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

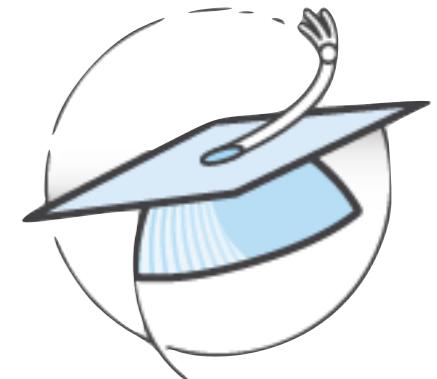
Squirrelbin, una app de ejemplo con lambda+DynamoDB creada por Amazon. Ejemplifica cómo implementar un API REST de tipo CRUD con BD



Probar los servicios FaaS

Por desgracia, la mayoría de estos servicios requieren de una tarjeta de crédito para darse de alta, pero hay algunas alternativas

- **Amazon Web Services:** la UA es un centro asociado al programa [AWS Educate](#). Podéis solicitar el alta en <https://www.awseducate.com/Application?apptype=student> (la cuenta tipo Starter no requiere tarjeta y tiene un crédito de 75\$)
- **Azure:** podéis probar las Azure Functions sin tener que daros de alta en <https://tryfunctions.com/try> (*se puede repetir las veces que se quiera, pero no se guardan los datos*)



Students

[Apply for AWS Educate for Students](#)

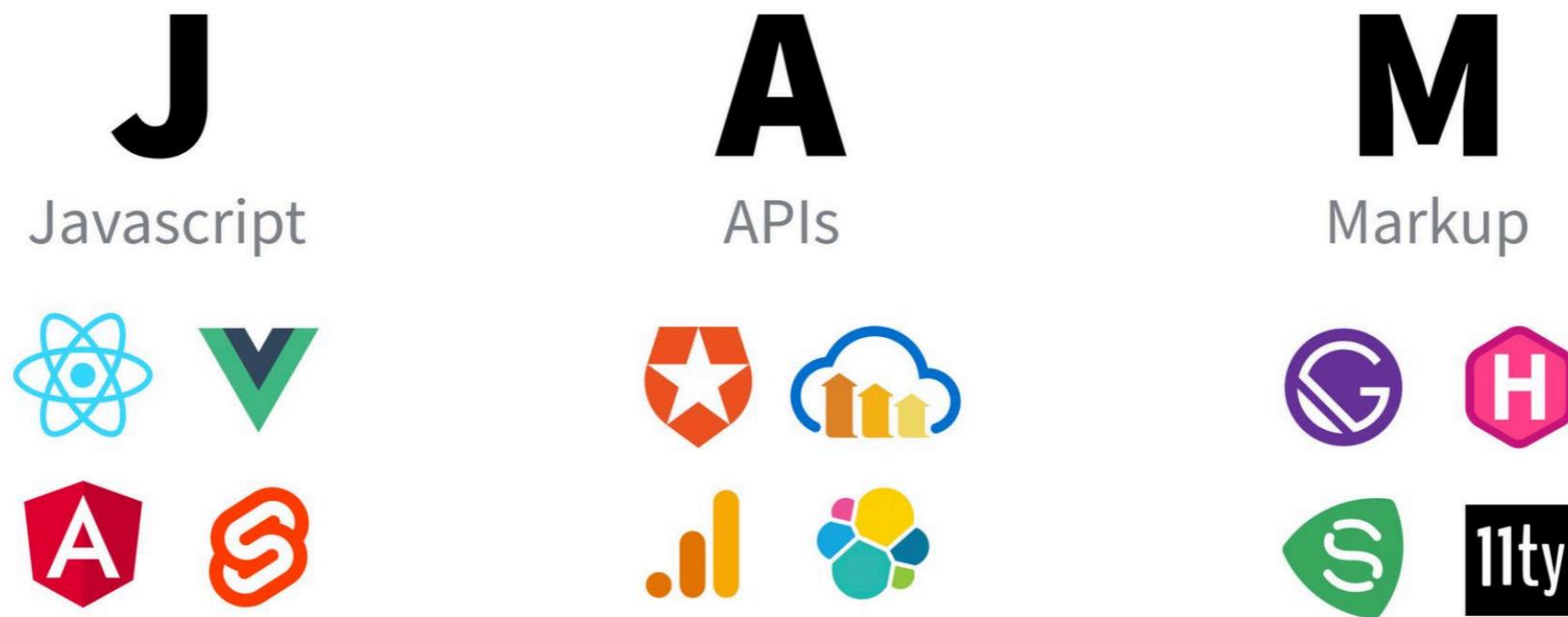
Tema 6: Aplicaciones web en “la nube”

5.4

Jamstack

JamStack

- Inicialmente una abreviatura de **Javascript**, **APIs** y **Markup**
 - Como actualmente todas las apps web usan Javascript, APIs y Markup, en realidad no es un nombre muy descriptivo 😳. Se ha dejado el nombre porque tomó cierta tracción, pero ya no “significa oficialmente” eso



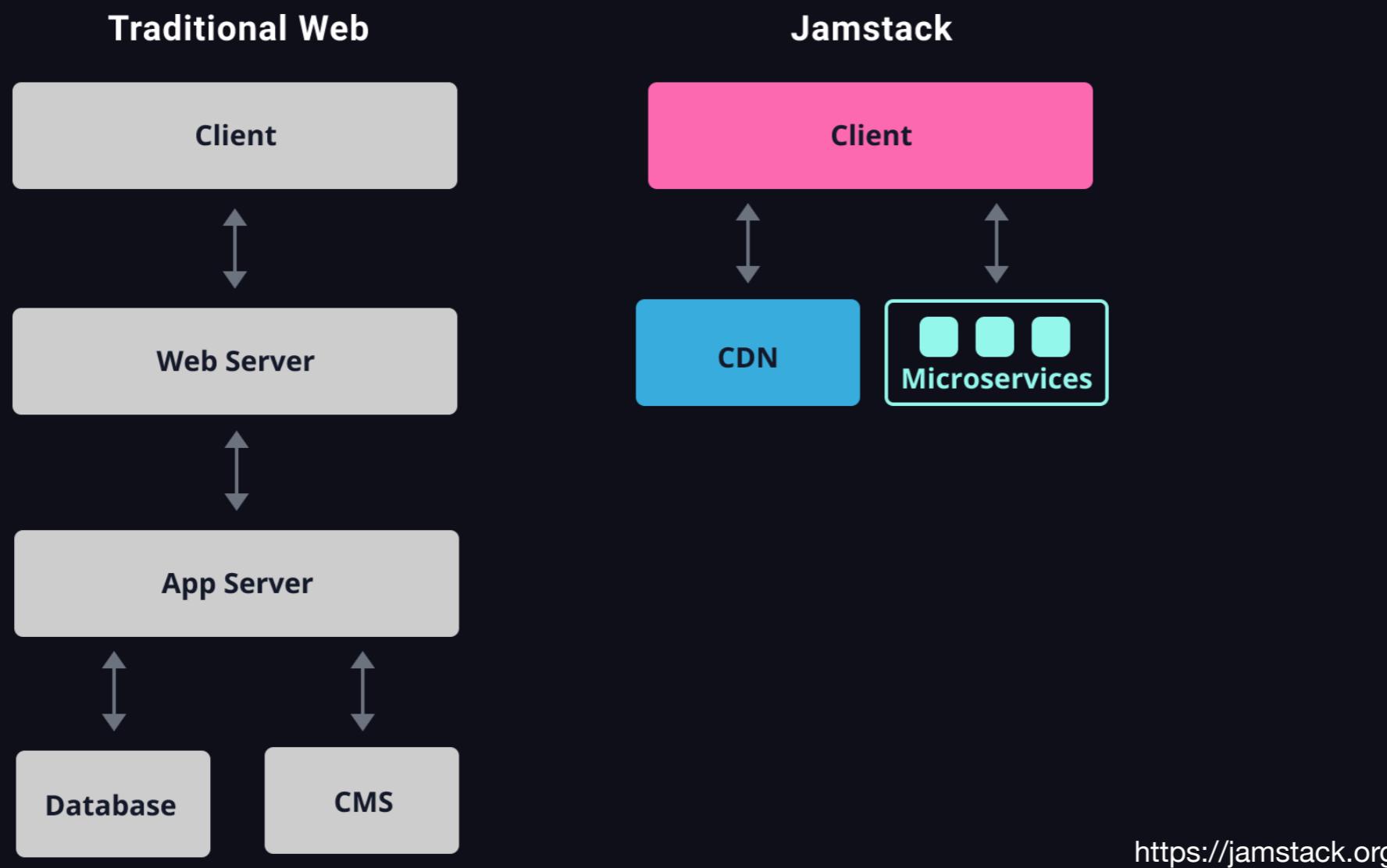
<https://www.freecodecamp.org/news/what-is-the-jamstack-and-how-do-i-host-my-website-on-it/>

JamStack vs. stack tradicional

The future is highly distributed

Jamstack is the new standard architecture for the web. Using Git workflows and modern build tools, pre-rendered content is served to a CDN and made dynamic through APIs and serverless functions.

Technologies in the stack include JavaScript frameworks, Static Site Generators, Headless CMSs, and CDNs.



El contenido web en JamStack

- En lugar de usar CMS “dinámicos” que unen en tiempo real la parte estática con la parte sacada de la base de datos, se usan **generadores estáticos** como Next.js (React), Nuxt (Vue), Hugo, Gatsby...

Entire Project on a CDN

Because Jamstack projects don't rely on server-side code, they can be distributed instead of living on a single server. Serving directly from a CDN unlocks speeds and performance that can't be beat. The more of your app you can push to the edge, the better the user experience.

Modern Build Tools

Take advantage of the world of modern build tools. It can be a jungle to get oriented in and it's a fast moving space, but you'll want to be able to use tomorrow's web standards today without waiting for tomorrow's browsers. And that currently means Babel, PostCSS, Webpack, and friends.

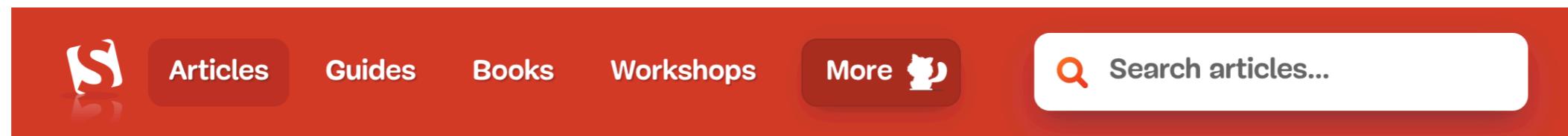
Automated Builds

Because Jamstack markup is prebuilt, content changes won't go live until you run another build. Automating this process will save you lots of frustration. You can do this yourself with webhooks, or use a publishing platform that includes the service automatically.

<https://jamstack.org/>

Ejemplo: Smashing Magazine

<https://www.smashingmagazine.com/2020/01/migration-from-wordpress-to-jamstack/>



JANUARY 28, 2020 • [31 comments](#)

How Smashing Magazine Manages Content: Migration From WordPress To JAMstack

ABOUT THE AUTHOR

Sarah Drasner is an award-winning Speaker, Head of Developer Experience at Netlify, Vue core team member, and Staff Writer at CSS-Tricks. Sarah is formerly ... [More about Sarah ...](#)

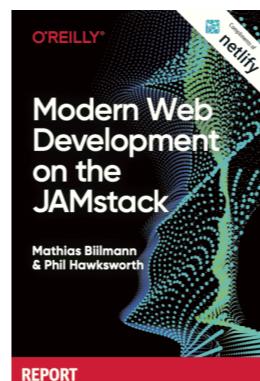
QUICK SUMMARY • WordPress adoption is massive. So why would a WordPress site consider moving to JAMstack? In this technical case study, we'll cover what an actual WordPress migration looks like, using Smashing Magazine itself! We'll talk through the gains and losses, the things we wish we knew earlier, and what we were surprised by.

11 min read

WordPress, Static Generators

Share on Twitter or LinkedIn

Saved for offline reading



<https://www.netlify.com/oreilly-jamstack/>

Tema 6: Aplicaciones web en “la nube”

5.5

Conclusiones

Ventajas

- **Simplificación** de la gestión de operaciones
- Escalabilidad
- Reducción de **costes**

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/>



Serverless Architectures

Serverless architectures are application designs that incorporate third-party “Backend as a Service” (BaaS) services, and/or that include custom code run in managed, ephemeral containers on a “Functions as a Service” (FaaS) platform. By using these ideas, and related ones like single-page applications, such architectures remove much of the need for a traditional always-on server component. Serverless architectures may

