

Status 426: Upgrade Required

# Módulo 5: Más allá de REST

---

# Motivación para el módulo de hoy

Evolución de los modelos de cómputo:

- Desde las apps monolíticas
- ... pasando por los sistemas operativos
- ... las bibliotecas/frameworks
- ... las aplicaciones distribuidas con RPC
- ... y la web como plataforma de aplicaciones mediante APIs REST

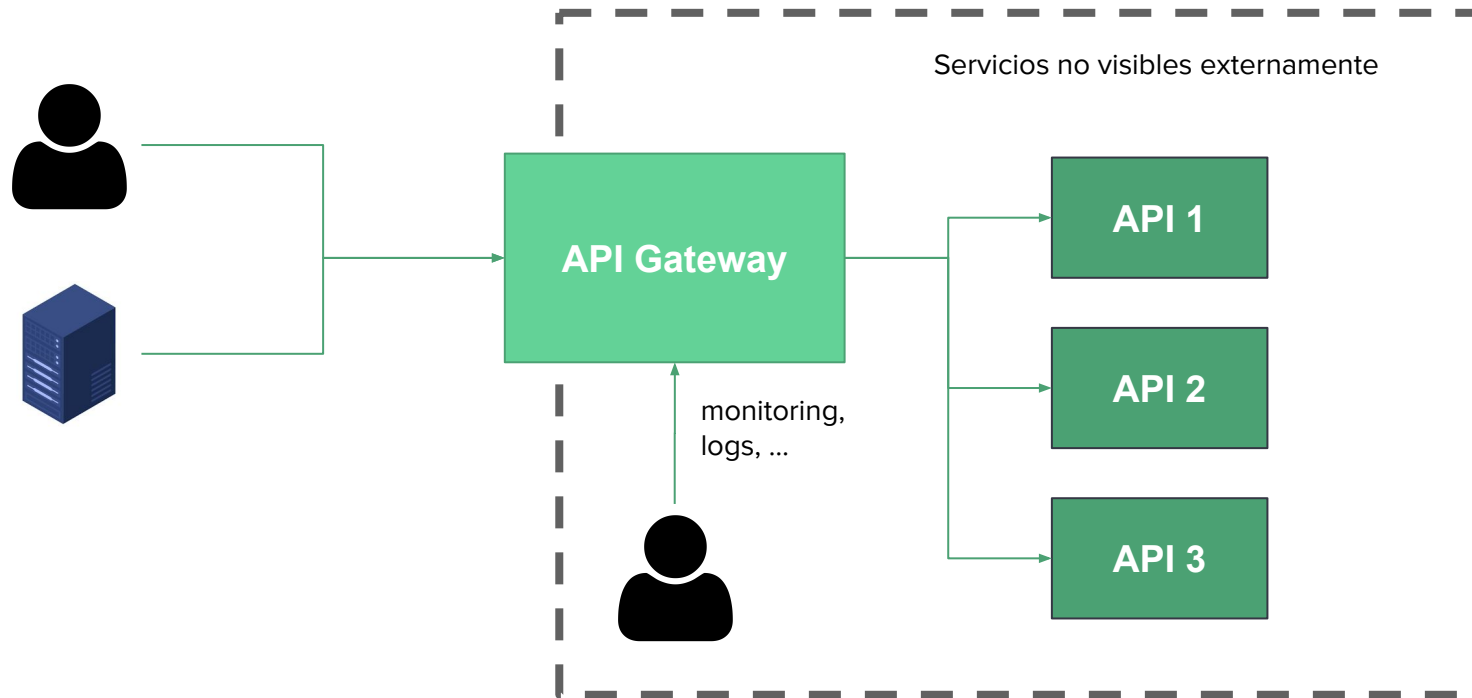
**En la clase de hoy:** ¿Ya está todo dicho? ¿Qué sigue después de REST?

# El problema: administrando muchas APIs

A medida que empezamos a tener decenas o cientos de APIs, se complica:

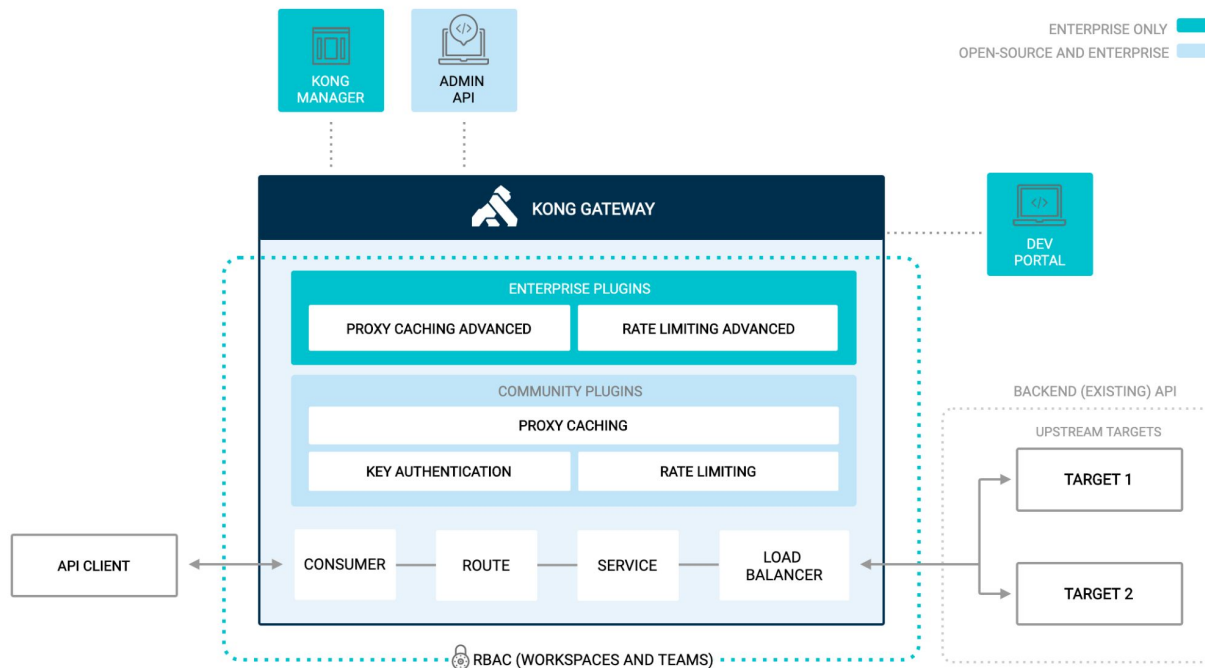
- Seguridad: ¿API keys para cada una?
- Logging: todo disjunto
- Policies: ¿cómo asegurar consistencia entre las distintas APIs?

# La solución: API Gateways



# Ejemplo: Kong API Gateway

Gateway Open Source lanzado en 2015



# Kong - DEMO

localhost:8001 interfaz administrativa (API REST)

localhost:8000 interfaz para procesar requests externos



# Los límites de REST

No todo lo que brilla es oro...

REST sufre particularmente

1. Performance en sistemas de  $\mu$ Servicios
2. Performance en conexiones de alta latencia
3. Imposibilidad de iniciar interacciones del lado del servidor

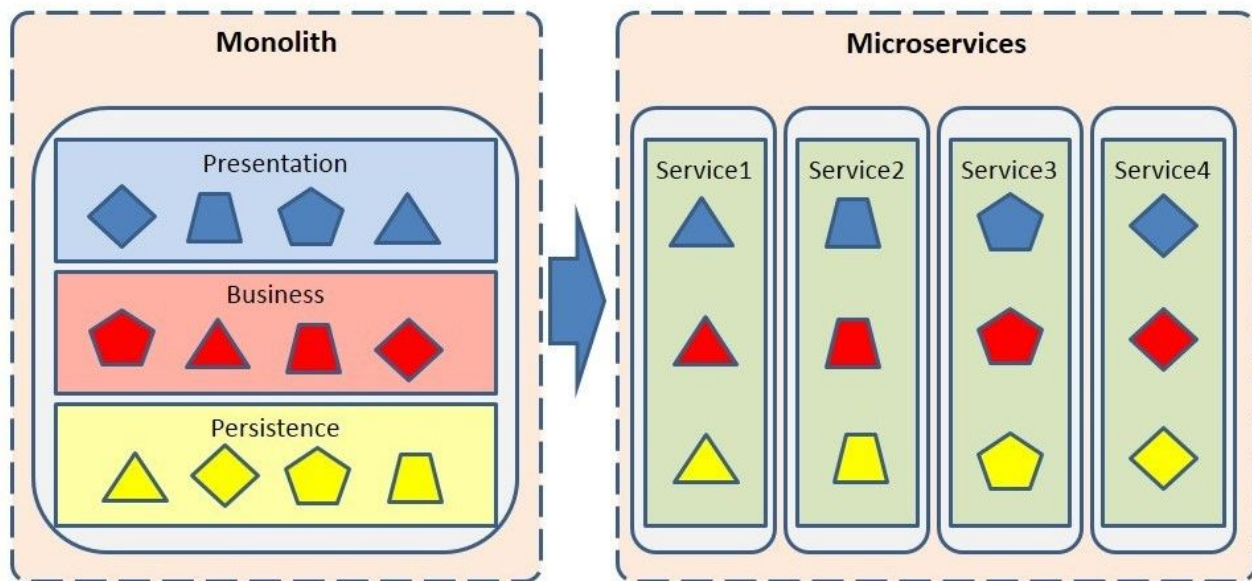


# Las apps ya no son monolíticas

Al tener muchos  $\mu$ Servicios, ¿cómo hacer para que se comuniquen entre sí?

REST brinda

- Overhead (headers, etc)
- No soporta async
- Timeouts

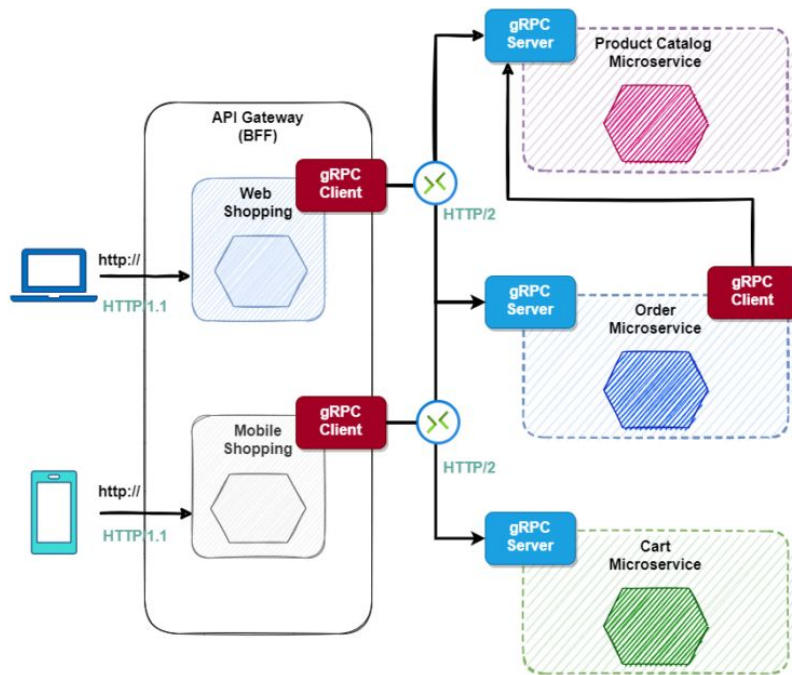




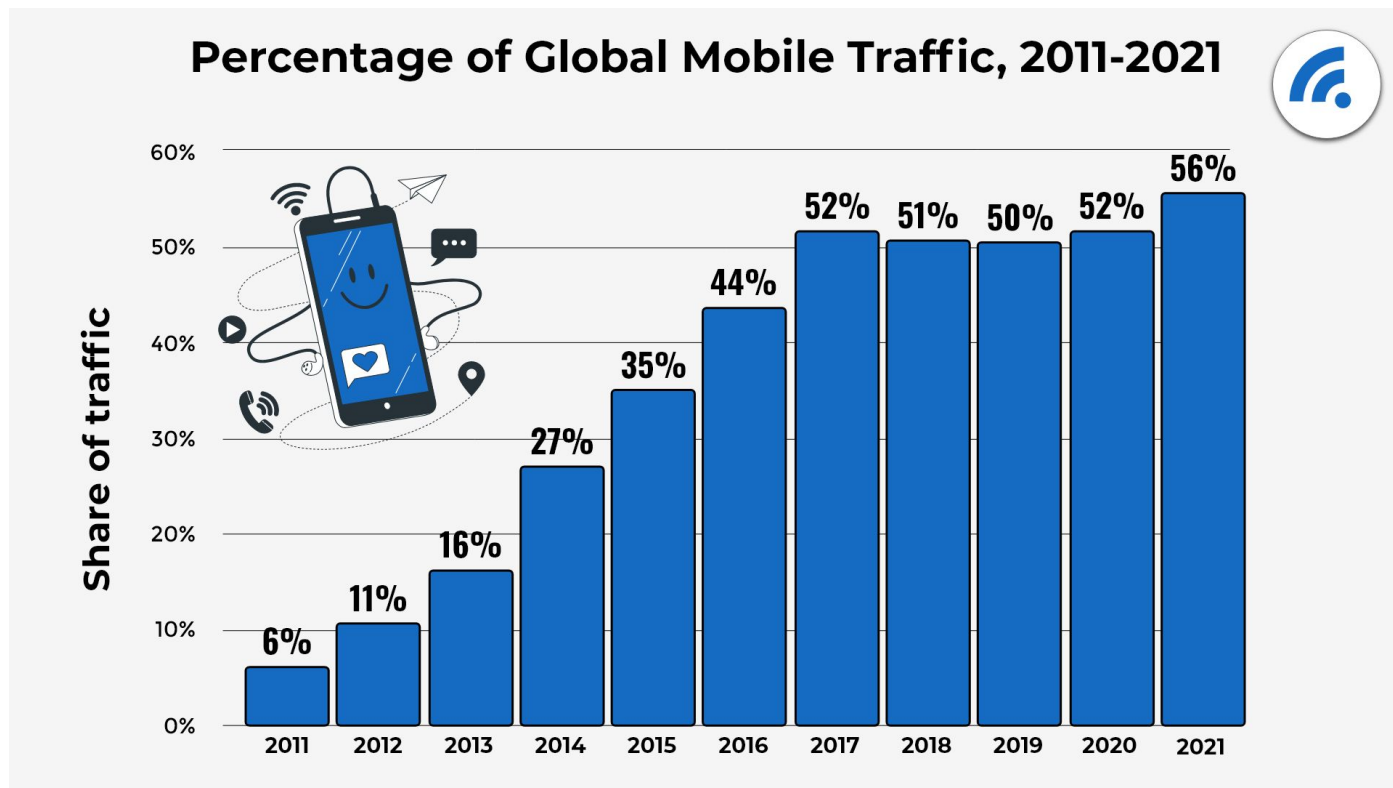
# Las apps ya no son monolíticas



Si la performance al comunicar  $\mu$ Services entre sí es crítica: RPC, por ej: gRPC + HTTP/2 o HTTP/3



# Las apps ahora corren en el celu



# Las apps ahora corren en el celu

**Desafío #1:** lidiando con conexiones de alta latencia

Ejemplo:

Armamos una app móvil para nuestra heladería

- Cuando el usuario va armando el pedido queremos mostrar un resumen de gustos por tipo
  - o Por ej: 2 chocolates, 1 crema, 2 frutas
- Pseudocódigo
  1. GET /pedidos/{pedidoId}/potes
  2. Para cada pote y cada gusto dentro de ese pote
    - a. GET /gustos/{gustoId}

A este problema se le llama ***1+N requests***.

- Es un problema en general. Pero en conexiones lentas es particularmente molesto

# GraphQL

Lanzado en 2015 por Facebook como una forma de obtener APIs más eficientes

Basado en HTTP pero no es RESTful. **No** usa status codes ni hypermedia ni verbos.

Permite explorar grafos y construir el resultado de forma explícita

## Request

```
{
  findPedido(id:1) {
    potes {
      gustos {
        tipo
      }
    }
  }
}
```

Un único request para resolver el problema 1+N

## Resultado

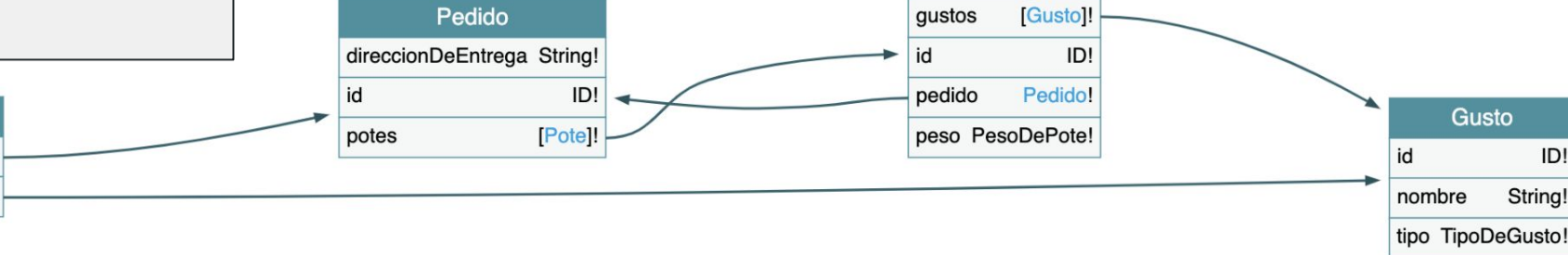
```
{
  "data": {
    "findPedido": {
      "potes": [
        {
          "gustos": [
            {
              "tipo": "DULCE_DE_LECHES"
            },
            {
              "tipo": "CHOCOLATES"
            }
          ]
        },
        {
          "gustos": [
            {
              "tipo": "DULCE_DE_LECHES"
            }
          ]
        }
      ]
    }
  }
}
```

Query	
findPedido	Pedido
gustos	[Gusto]!

Pedido	
direccionDeEntrega	String!
id	ID!
potes	[Pote]!

Pote	
gustos	[Gusto]!
id	ID!
pedido	Pedido!
peso	PesoDePote!

Gusto	
id	ID!
nombre	String!
tipo	TipoDeGusto!



# GraphQL - DEMO



<https://curso-apis-production.up.railway.app/guidodecaso/heladeria/1.0.0/graphql>

# GraphQL: mutaciones

La **Q** de GraphQL es por **Query** pero sin embargo soporta **mutaciones**

## Spec

```
type Mutation {  
  agregarPote(pedidoId: ID!, peso: PesoDePote!, gustoIds: [ID]!) : Pote!  
}
```

## Request

```
mutation {  
  agregarPote(pedidoId: 1, peso: _1000, gustoIds: ["choco_am"]) {  
    id  
    gustos {  
      nombre  
    }  
  }  
}
```

## Resultado

```
{  
  "data": {  
    "agregarPote": {  
      "id": "7",  
      "gustos": [  
        {  
          "nombre": "Chocolate amargo"  
        }  
      ]  
    }  
  }  
}
```

# Las apps ahora corren en el celu

**Desafío #2:** iniciar actividad desde el servidor, por ejemplo notificaciones

Ejemplo:

Supongamos que nuestra app móvil quiere notificar al usuario cuando el delivery está por llegar

¿Cómo hacer esto con una API REST siendo que el cliente siempre es el que inicia la interacción?

Una posible (mala) solución es que el cliente haga polling como en el mecanismo POST/GET/GET

A este problema se le llama ***Server initiated interactions***.

- En general las APIs REST no ofrecen una solución elegante a este problema.

# GraphQL: server initiated interactions

Un mecanismo completamente distinto al de consulta, ya que el resultado viaja vía **WebSockets**.

## Spec

```
type Subscription {  
  lanzamientoDeGustoNuevo : Gusto!  
  pedidoEstaProntoALlegar(id: ID) : Pedido!  
}
```

## Resultado

## Request

```
subscription EscucharGustosNuevos {  
  lanzamientoDeGustoNuevo {  
    id  
    nombre  
  }  
}
```

```
{  
  "data": {  
    "lanzamientoDeGustoNuevo": {  
      "id": "crema_cielo",  
      "name": "Crema del cielo"  
    }  
  }  
}
```



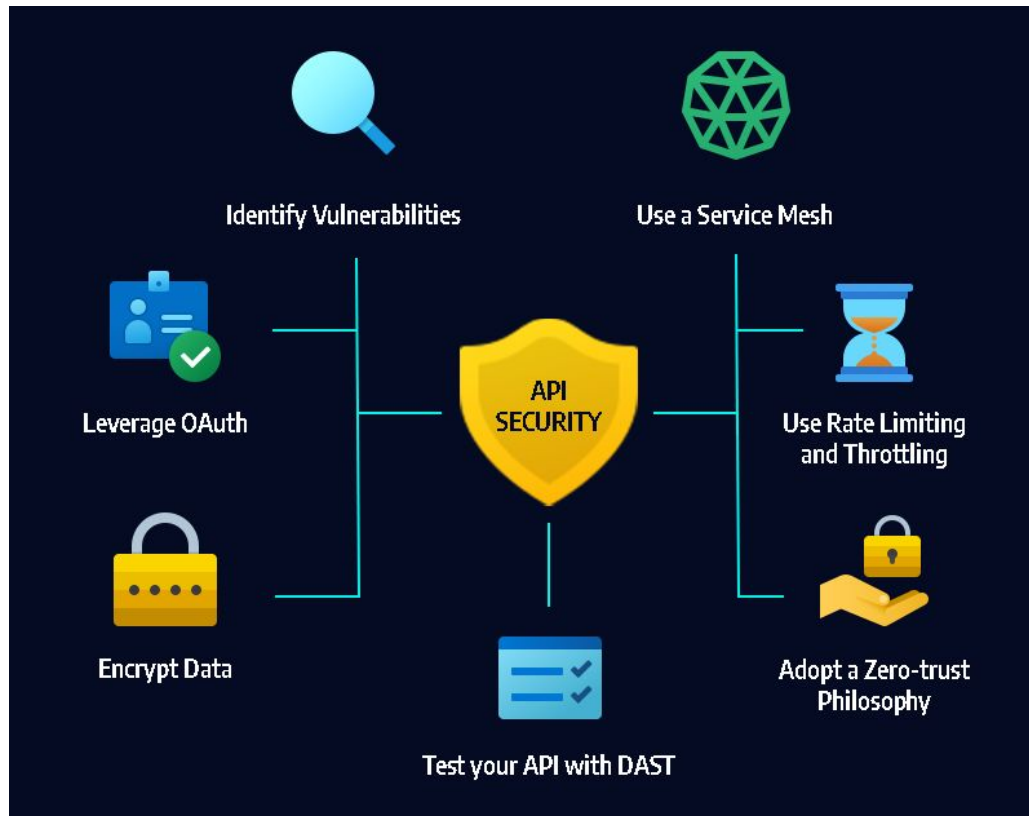
# APIs: REST vs. GraphQL

	REST	GraphQL
Payload a medida del cliente	✗	✓
Mutación mediante verbos estándar	✓	✗
Soporte nativo de introspección	✗	✓
Favorece caching	✓	✗
Tipado fuerte	✗	✓
Soporte para <i>server-initiated interactions</i>	✗	✓

Conclusión: depende para qué lo necesitemos, un estilo puede ser más útil que otro según el contexto

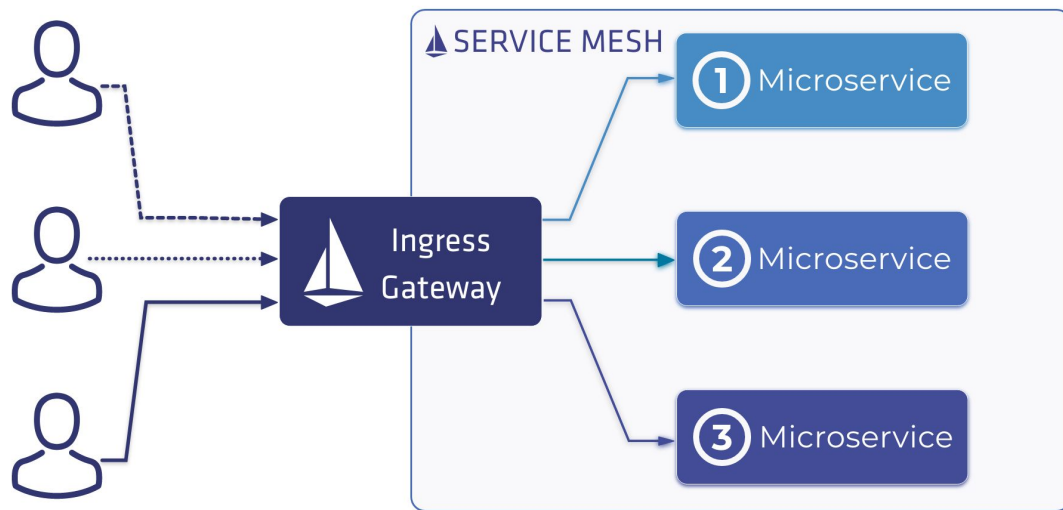
# El futuro de las APIs

**Predicción 1.** Mayor foco en la seguridad de las APIs.



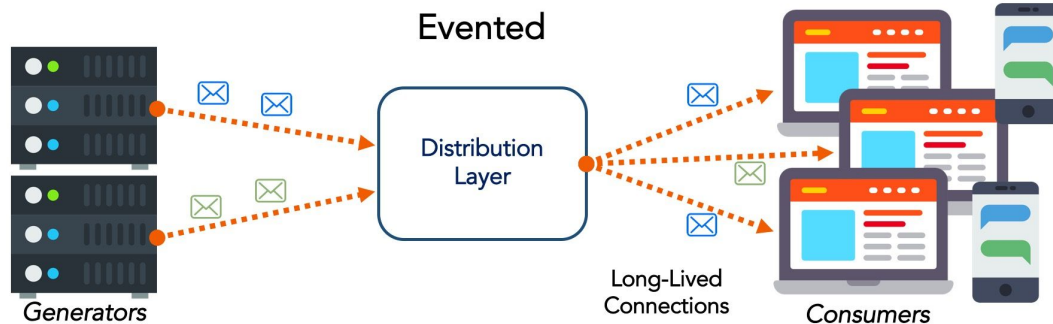
# El futuro de las APIs

**Predicción 2.** Surgimiento de patrones arquitectónicos para APIs de  $\mu$ Services.



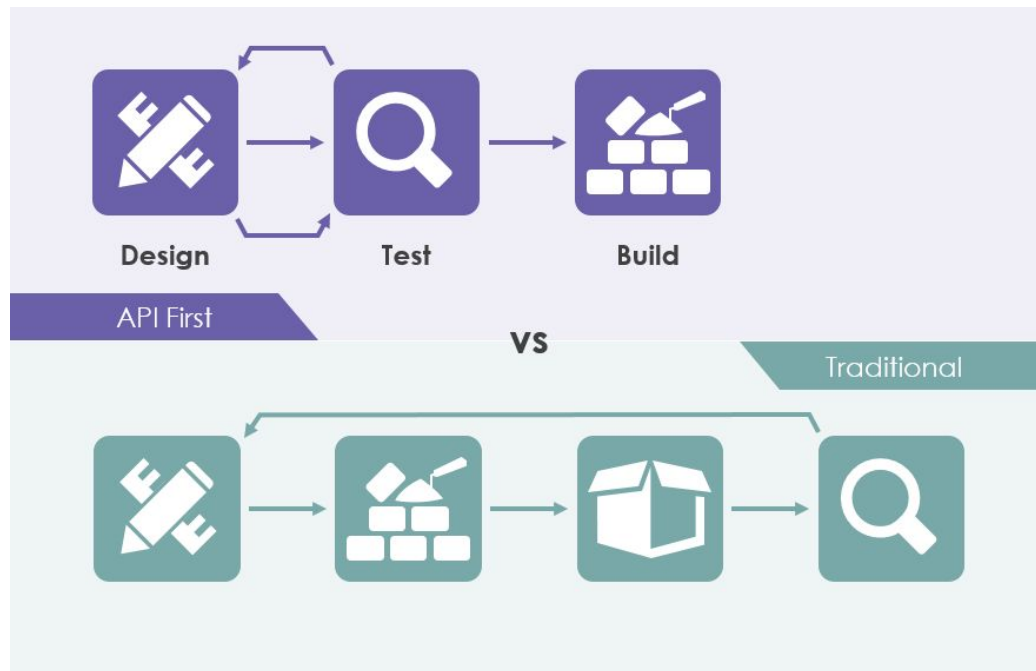
# El futuro de las APIs

**Predicción 3.** Resurgimiento de las APIs orientadas a eventos.



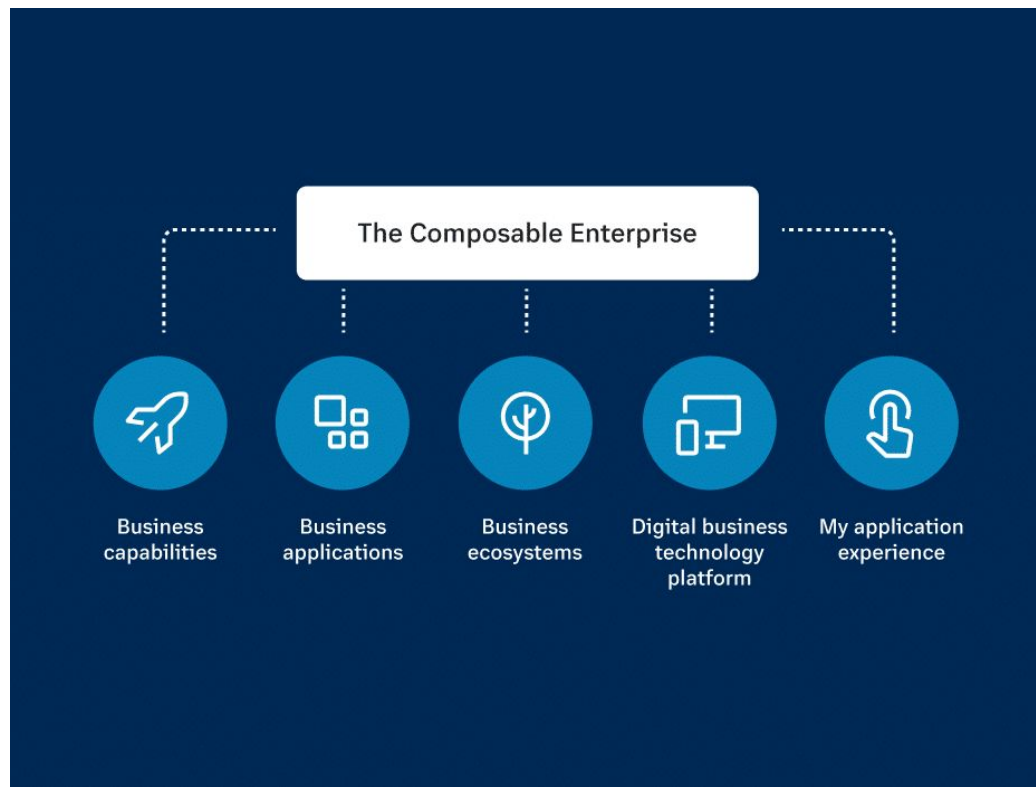
# El futuro de las APIs

**Predicción 4.** API-first será la norma y no la excepción.



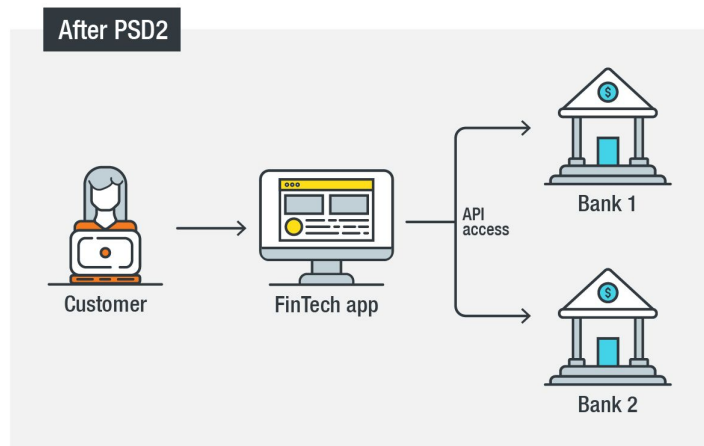
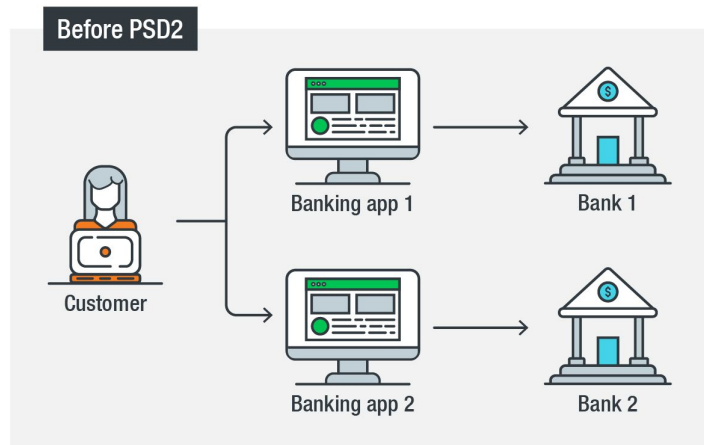
# El futuro de las APIs

**Predicción 5.** APIs como forma de comunicación intraempresa.  
Automation y no-code.



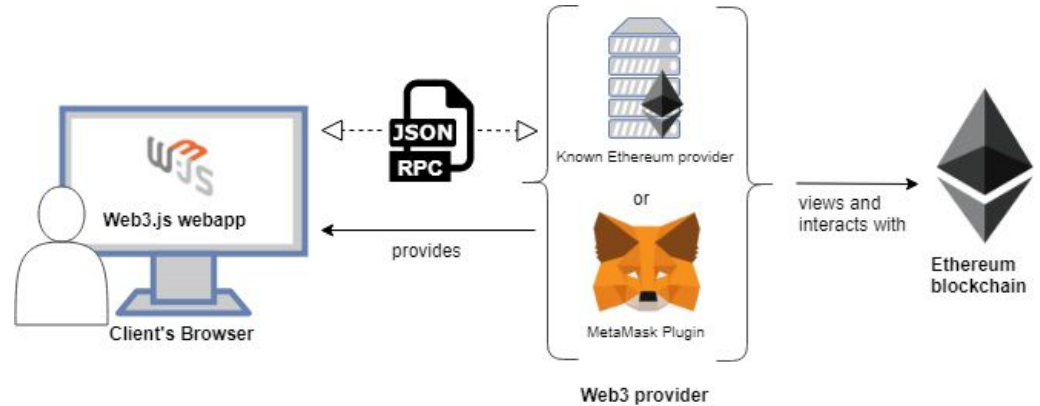
# El futuro de las APIs

**Predicción 6.** API standards, governance y control.






# El futuro de las APIs

**Predicción 7.** Surgimiento de modelos teóricos para las APIs en Smart Contracts (web3/crypto).





# Conclusiones

- No subestimen la importancia de diseñar una API
  -  adopción,  developer experience,  revenue
- Apliquen la metodología API-first
  - Desacoplar equipos, feedback temprano
- Pensar qué tipo de API es la que mejor se adapta a cada contexto
  - RPC, GraphQL, REST, Evented, ...
- Abrir APIs es “fácil”, cerrarlas es **imposible**
  - Empezar con los mínimos datos necesarios, agregar datos u operaciones es fácil
- Si van a exponer muchas APIs, invertir en un proceso
  - Governance, API guidelines, portal unificado, tipos reusables, fragmentos

Status 418: I'm a Teapot

¿Preguntas?

---

Status 410: Gone

¡Muchas gracias!

---