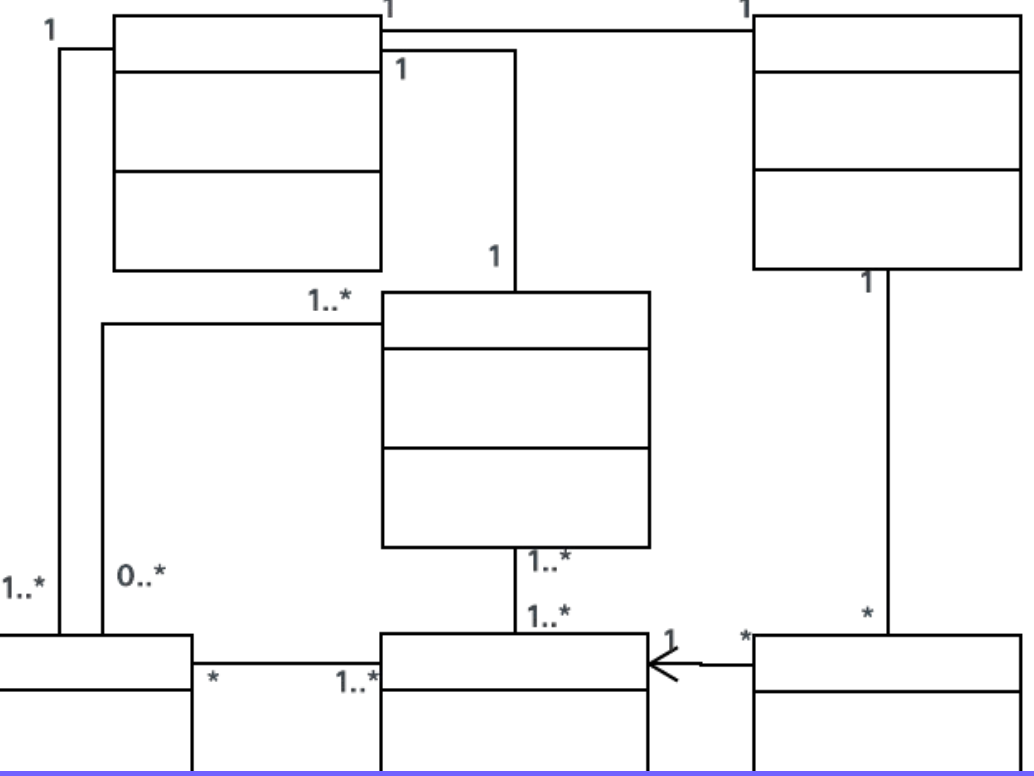
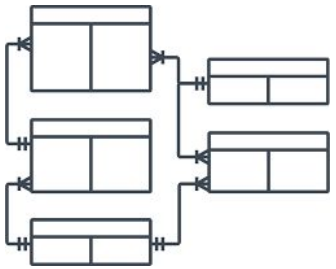




Diseño de Sistemas

Agenda

- Integración de Sistemas

  - Tipos de Integración
  - Aplicaciones Tradicionales vs Nube
  - Algunos Componentes Arquitectónicos

Integración de Sistemas

¿Qué es integrar Sistemas?

- Integrar sistemas es establecer una *comunicación* entre ellos.
- Es *compartir* datos y funcionalidades entre dos o más sistemas.
- Es *abrir el universo* "cerrado" sobre el que fue pensado y diseñado un sistema.
- Es *aumentar la cohesión* de un sistema evitando duplicar lógicas existentes.

Integración de Sistemas

¿Por qué integrar sistemas?

Integración de Sistemas

Necesidades de Integración

- Los sistemas, generalmente, no son aislados y necesitan de otros para subsistir.
- Existen lógicas que ya están creadas que resuelven ciertas tareas.
- Los sistemas deben compartir datos entre sí y no desean “copiarlos” sino “consultarlos”.
- Necesidades vinculadas al negocio.
- Necesidades vinculadas al marco legal.

Integración de Sistemas

¿Sincrónica o Asincrónica?

Esta decisión dependerá exclusivamente de las necesidades del negocio y/o restricciones técnicas.

Integración de Sistemas

Ejemplos

Necesidad de Negocio

Se tiene un E-Commerce que necesita procesar pagos a través de la plataforma.

Necesidades vinculadas al marco legal

Se tiene un sistema que necesita realizar facturas electrónicas conforme a la reglamentación legal actual de Argentina.

Integración de Sistemas - Ejemplo

Ejemplo: *Sistema de Telepase*

- ¿Qué es un Sistema de Telepase?
- ¿Qué actores involucra?
- ¿Qué Procesos involucra?
- ¿Qué Tecnologías involucra?
- ¿Qué Integraciones Involucra?

Integración de Sistemas - Ejemplo

Sistema de Telepase - ¿Qué es un Sistema de Telepase?

Es un Sistema que te permite ahorrar tiempo, utilizando los distintos peajes de la Red de Autopistas, sin detener el vehículo. Funciona mediante un dispositivo electrónico llamado TAG, el cual es colocado en el parabrisas del vehículo y al traspasar los peajes de las vías de TelePASE de las distintas autopistas, es captado por una antena que produce la apertura automática de la barrera.

Integración de Sistemas - Ejemplo

Sistema de Telepase - Análisis

¿Qué actores involucra?
Usuario, Sistema de Peaje, Sistema de Pago, Instalador

¿Qué Procesos involucra?
Alta de Usuario, Vinculación Oblea/Usuario, Detección de Pasada, Cobro Mensual, Desvinculación Usuario

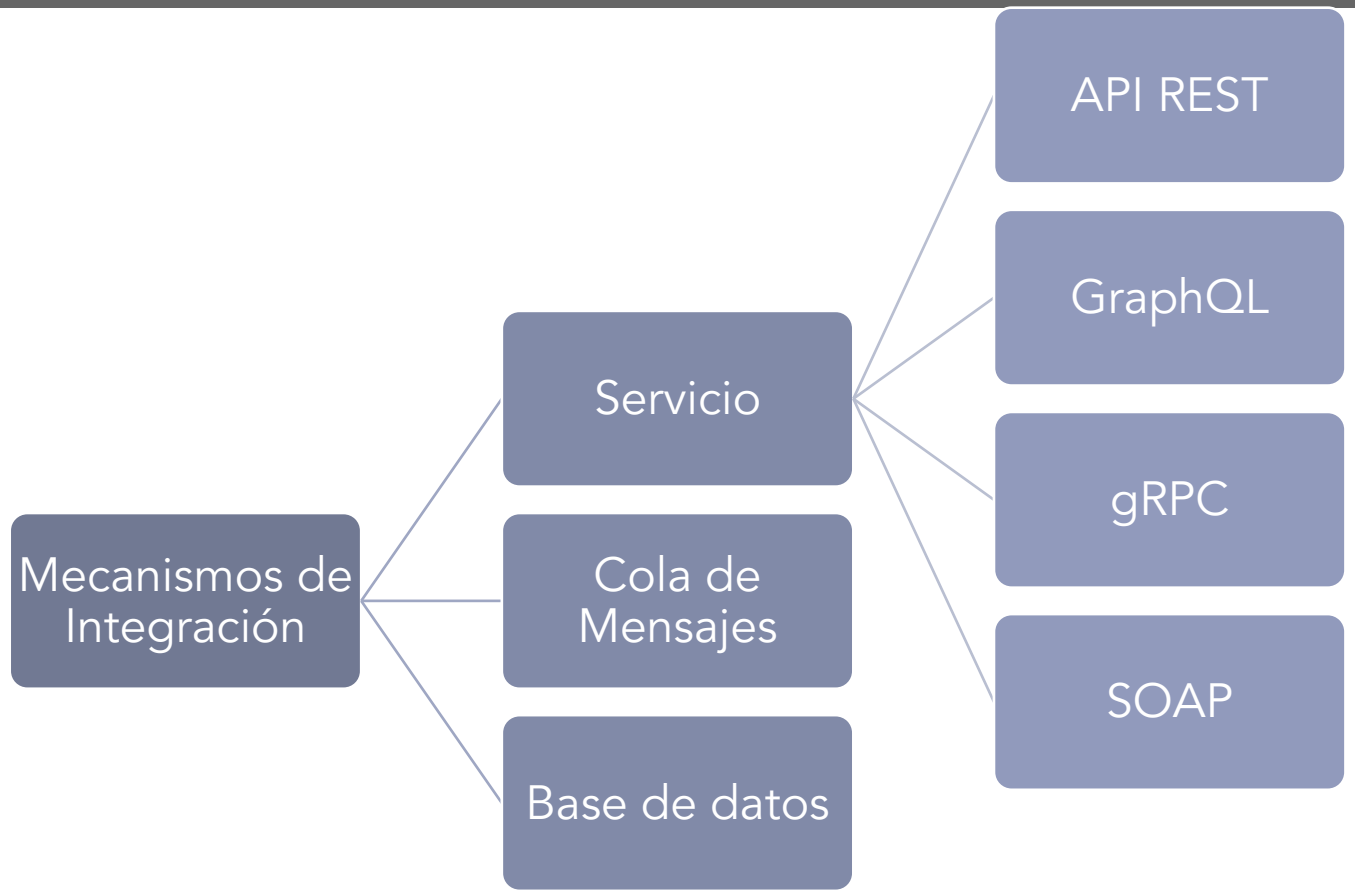
¿Qué Tecnologías involucra?
RFID, Arquitectura Web, Mobile

Integración de Sistemas - Ejemplo

Sistema de Telepase - ¿Qué integraciones involucra?

- Sincronización Cabinas de Peaje <-> Unidad Central
- Unidad Central -> Gateway de Pago (Generación Pagos)
- Gateway de Pago -> Unidad Central (Bajas por Falta de Pago)
- Gateway de Pago <-> Tarjeta de Crédito
- Tarjeta de Crédito <-> Entidad Bancaria

Algunos Mecanismos de Integración



¿Qué es un Web Service?

*Un servicio web es un **término genérico** para un **componente de software interoperable** de **máquina a máquina** que está **alojado** en una **ubicación direccionable de red**.*

- *Posee una interfaz*
- *Oculta los detalles de implementación para que se pueda utilizar independientemente de la plataforma y del stack tecnológico.*
- *Se busca implementaciones de tecnología cruzada, orientadas a componentes y poco acopladas.*
- *Pueden utilizar solos o con otros servicios web para realizar una agregación compleja o una transacción comercial.*

¿Qué es una API?

Una interfaz de programación de aplicaciones (API) es un conjunto de herramientas, definiciones y protocolos que se usa para diseñar e integrar aplicaciones.

Permite que un producto o servicio se comunique con otros productos y servicios, sin la necesidad de saber cómo se implementan internamente.

Tipos de API (Arquitectura)

- Local
  - Proporciona Servicios de Sistema Operativo, Bibliotecas o Middleware.
- Web
  - Proporciona Servicios entre Componentes Distribuidos y en general sobre el protocolo HTTP.
- Programa
  - Proporciona la posibilidad de Ejecución de Código de forma Remota entre Componentes Distribuidos (RPC).

Tipos de API (Alcance)

- Public (Pública)
- Partner (Para Terceros)
- Private (Interna)

Tipos de API (Web)

- REST
- SOAP
- gRPC
- GraphQL

REST

NO es un framework.

NO es una tecnología.

NO es un patrón de diseño.

NO es un protocolo.

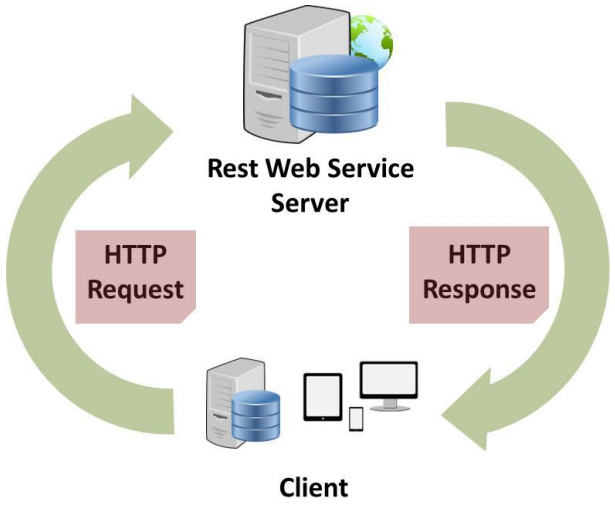
NO es un estándar.

REST

REST o Representational State Transfer es un ESTILO **O TÉCNICA** de Arquitectura a la hora de realizar **una comunicación entre cliente y servidor.**

Se apoya sobre el Protocolo HTTP

Crea una solicitud HTTP que contiene toda la información necesaria, es decir, **un REQUEST a un servidor tiene toda la información necesaria y solo espera una RESPONSE**, ósea una respuesta en concreto.



REST

- Se apoya en los métodos básicos de HTTP, como son:
  - **Post**: Para crear recursos nuevos.
  - **Get**: Para obtener un listado o un recurso en concreto.
  - **Put**: Para modificar un recurso.
  - **Patch**: Para modificar un recurso (en forma parcial).

Delete: Para borrar un recurso.

404

Page not found

The Page you are looking for doesn't exist or an other error occurred.

Go back, or head over to weeblr.com to choose a new direction.

REST – Códigos de Estado HTTP

- 1xx: Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.
- 2xx: Respuestas correctas. Indica que la petición ha sido procesada correctamente.
- 3xx: Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.
- 4xx: Errores causados por el cliente. Indica que ha habido un error en el procesado de la petición a causa de que el cliente ha hecho algo mal.
  5xx: Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

REST – Características

- Todos los objetos se manipulan mediante URI (identificador uniforme de recursos)
- Utiliza como formato de intercambio JSON / XML

SOAP

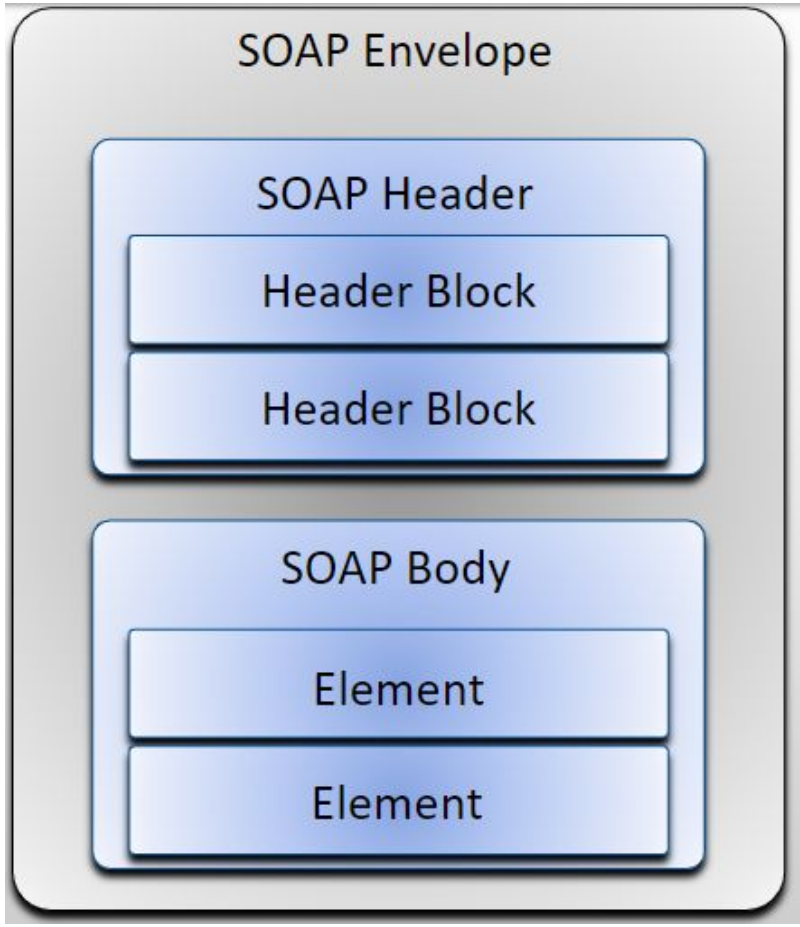
- Los servicios SOAP (Web Services), son servicios que basan su comunicación bajo el protocolo SOAP (Simple Object Access Protocol).

- SOAP es un protocolo estándar que define cómo dos componentes pueden comunicarse por medio de intercambio de datos XML.

- Funcionan por lo general a través del protocolo HTTP que es lo más común cuando invocamos un Web Services, sin embargo, SOAP no está limitado a este protocolo, si no que puede ser enviado por FTP, POP3, SMTP, TCP, Colas de mensajería.

- Utiliza WSDL (Web Services Description Language) que se usa para generar una interfaz del Servicio Web, misma que será su punto de entrada. A éste se le conoce como Archivo wsdl o Contrato, éste archivo estará accesible en una red por medio de un url que tiene terminación ?wsdl.

SOAP - ESTRUCTURA




```
<?xml version="1.0"?>

<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap
-envelope"
soap:encodingStyle="http://www.w3.org/2001
/12/soap-encoding">

     <soap:Header>
     ...
     </soap:Header>

     <soap:Body>
     ...
      <soap:Fault>
      ...
      </soap:Fault>

     </soap:Body>

</soap:Envelope>
```

SOAP - ESTRUCTURA

Solicitud

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   </soap:Header>
   <soapenv:Body>
      <aplicarVentaRequest xmlns="http://elements.blogsoa.com/Productos">
         <codigoProducto>1KL979GY</codigoProducto>
         <cantidadProducto>2</cantidadProducto>
         <almacen>Centro</almacen>
         <precioTotal>7600.72</precioTotal>
      </aplicarVentaRequest>
   </soapenv:Body>
</soapenv:Envelope>
```

Respuesta

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Header xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   </soap:Header>
   <soapenv:Body>
      <aplicarVentaResponse xmlns="http://elements.blogsoa.com/Productos">
         <codigo>0</codigo>
         <descripcion>La venta se ha registrado exitosamente</descripcion>
         <numeroOrden>356-MNC298</numeroOrden>
      </aplicarVentaResponse>
   </soapenv:Body>
</soapenv:Envelope>
```

SOAP - WDSL

En el archivo **wsdl** se **describe** todo el **servicio web**, las **operaciones disponibles** y las **estructuras de datos** de los **mensajes** de cada **operación**(xsd).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions name="Productos"
        targetNamespace="http://services.blogsoa.com/Productos"
        xmlns:client="http://services.blogsoa.com/Productos"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:prod="http://elements.blogsoa.com/Productos"
        xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
        xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>
    <schema xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://elements.blogsoa.com/Productos" schemaLocation="../XSD/productos.xsd"/>
    </schema>
  </wsdl:types>

  <!-- Consultar Productos Message -->
  <wsdl:message name="consultarProductosRequest">
    <wsdl:part name="parameters" element="prod:consultarProductosRequest"/>
  </wsdl:message>
  <wsdl:message name="consultarProductosResponse">
    <wsdl:part name="parameters" element="prod:consultarProductosResponse"/>
  </wsdl:message>
```

SOAP - XSD

En el **archivo xsd** se describen las **estructuras de datos** de los **mensajes** de cada **operación**.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    elementFormDefault="qualified"
    targetNamespace="http://elements.blogsoa.com/Productos"
    xmlns:tipos="http://elements.blogsoa.com/Productos"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" >

<!-- Consultar la lista de Productos disponibles -->
<xsd:element name="consultarProductosRequest" type="tipos:ConsultarProductosRequest"/>
<xsd:element name="consultarProductosResponse" type="tipos:ConsultarProductosResponse"/>

<!-- Aplicar Venta de un Producto -->
<xsd:element name="aplicarVentaRequest" type="tipos:AplicarVentaRequest"/>
<xsd:element name="aplicarVentaResponse" type="tipos:AplicarVentaResponse"/>


<xsd:complexType name="ConsultarProductosRequest">
 <xsd:sequence>
  <xsd:element minOccurs="0" maxOccurs="1" name="codigoProducto" type="xsd:string"/>
  <xsd:element minOccurs="0" maxOccurs="1" name="nombreProducto" type="xsd:string"/>
  <xsd:element minOccurs="0" maxOccurs="1" name="categoria" type="xsd:string"/>
  <xsd:element minOccurs="0" maxOccurs="1" name="almacen" type="xsd:string"/>
 </xsd:sequence>
</xsd:complexType>
```

REST VS SOAP

Característica	REST	SOAP
Formato de Intercambio	XML / JSON	XML
Llamada/Respuesta	Sincrónica	Asincrónica/Sincrónica
Interfaz	Fijas y Simples	Dinámicas y Complejas
Descripción del Servicio	Documentación	WSDL
Acoplamiento	Bajo	Alto
Protocolos	HTTP	HTTP / SMTP / TCP
Seguridad	HTTPS	WS-Security
Manejo de Estados	Sin Estados (Stateless)	Con Estado
Tipo de Aplicación	Web - Mobile	Enterprise
Público	General	Específico

gRPC

gRPC es un framework de llamada a procedimiento remoto (RPC) moderno de código abierto de alto rendimiento que puede ejecutarse en cualquier entorno.

Puede conectar servicios de manera eficiente en y entre centros de datos con soporte para balanceo de carga, rastreo, verificación de estado y autenticación.

También es aplicable para conectar dispositivos, aplicaciones móviles y navegadores a servicios de backend.

gRPC

- Desarrollado por **Google** en el año 2015 (Cloud Native Computing Foundation).
- El transporte de **flujos de datos** mediante **HTTP/2**.
- Gestiona la **estructura y la distribución** de los datos mediante los **Protocol buffers** (archivos de texto plano con la extensión **.proto**)
- Los **flujos transmiten** en **datos binarios compactos** que surgen en la **serialización** y la **deserialización**.

gRPC - Protobuf

- Protocol Buffers (abreviado como Protobuf) es un formato de intercambio de datos desarrollado originalmente para uso interno en Google y lanzado al gran público por la empresa en 2008 como proyecto de código abierto.

- Este formato binario permite a las aplicaciones almacenar e intercambiar datos estructurados fácilmente, incluso si los programas están compilados en diferentes lenguajes.

gRPC - Protobuf

```
syntax = "proto3";

package tutorial;

option java_package = "com.example.tutorial";
option java_outer_classname = "AddressBookProtos";

message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phones = 4;
}

message AddressBook {
  repeated Person people = 1;
}
```

required: indica que es obligatorio asignar un valor al campo.

optional: en los campos opcionales, es posible indicar un valor, aunque no es necesario.

repeated: los campos con este modificador pueden repetirse un número indefinido de veces (incluso ninguna).

gRPC – Pasos de Implementación

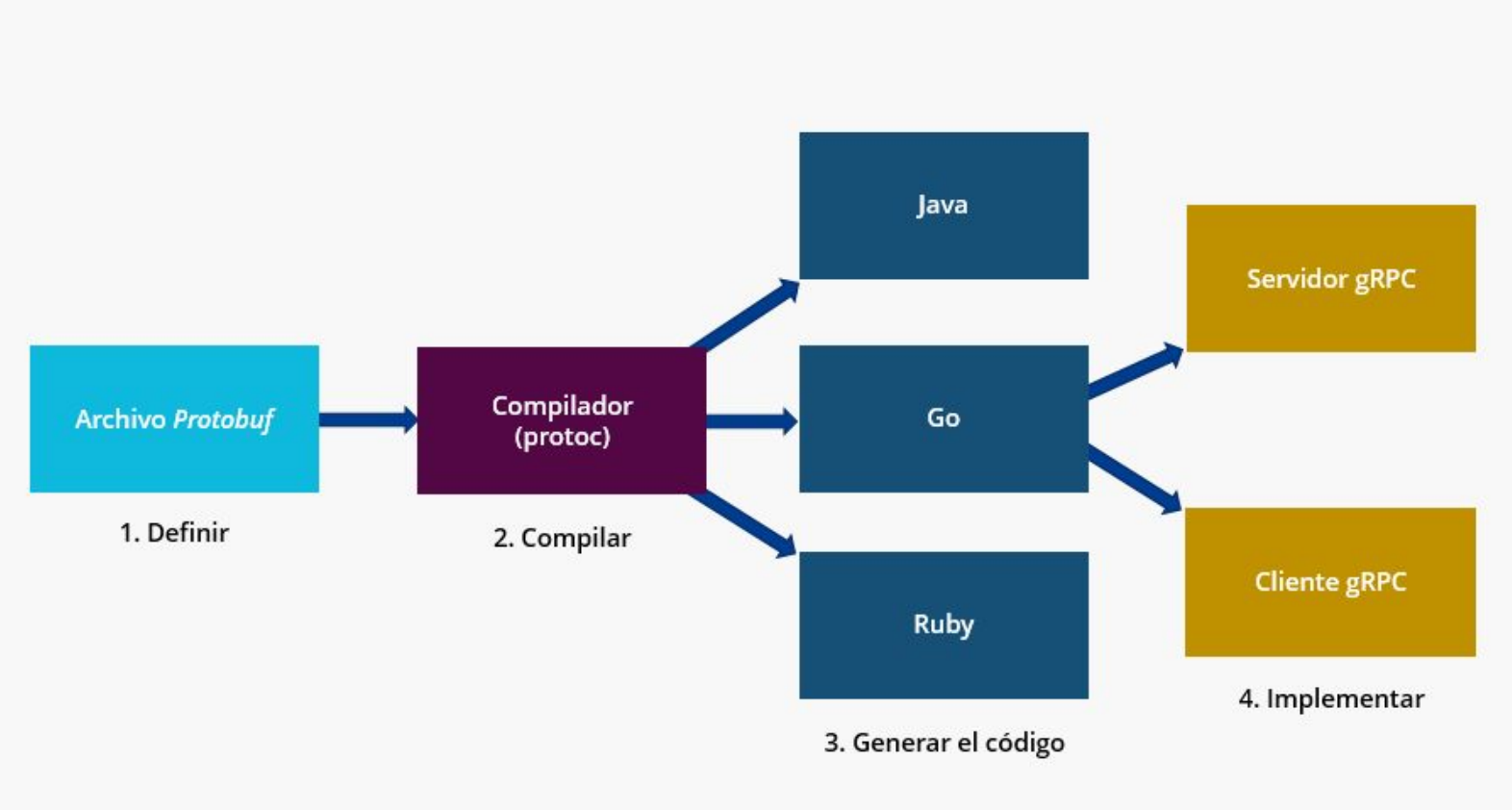
1.Definición del contrato de servicio

2.Generación del código gRPC del archivo *.proto (en el lenguaje deseado).*

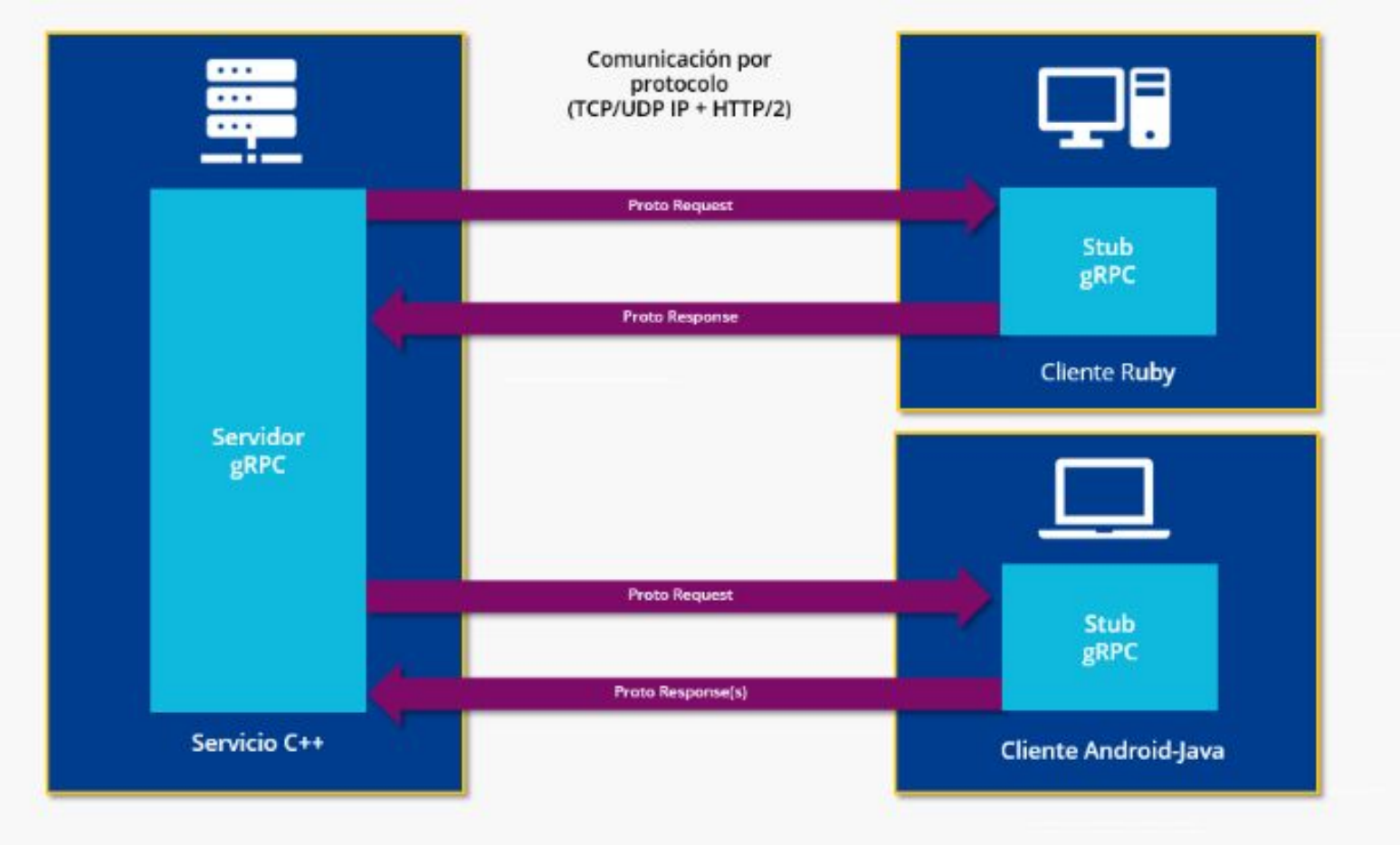
3.Implementación del servidor en el lenguaje deseado.

4.Creación del *stub* del cliente que llama al servicio.

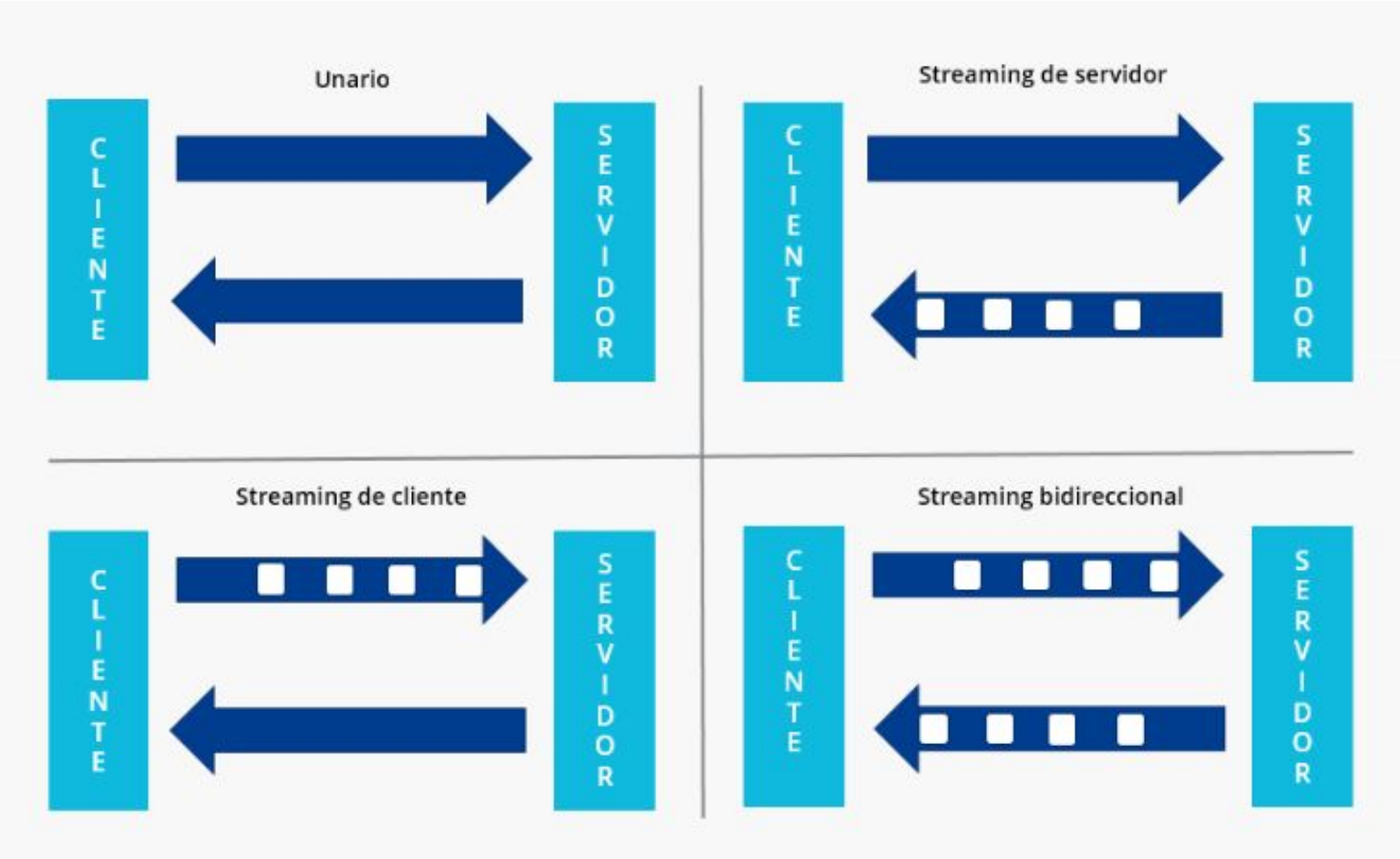
gRPC – Pasos de Implementación



gRPC – Arquitectura



gRPC – Arquitectura



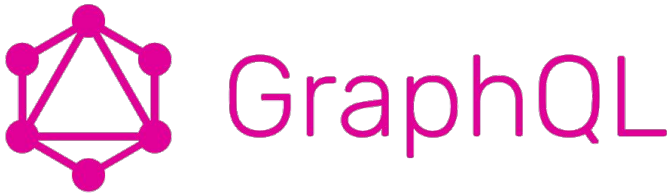
gRPC – Conclusiones

- Centrado en la Comunicación entre Servicios/Microservicios
- Soporte Multilenguaje
- Flujo de Intercambio más pequeño
- Streaming Bidireccional

GraphQL

Es un lenguaje de consulta para API y un tiempo de ejecución para completar esas consultas con sus datos existentes.

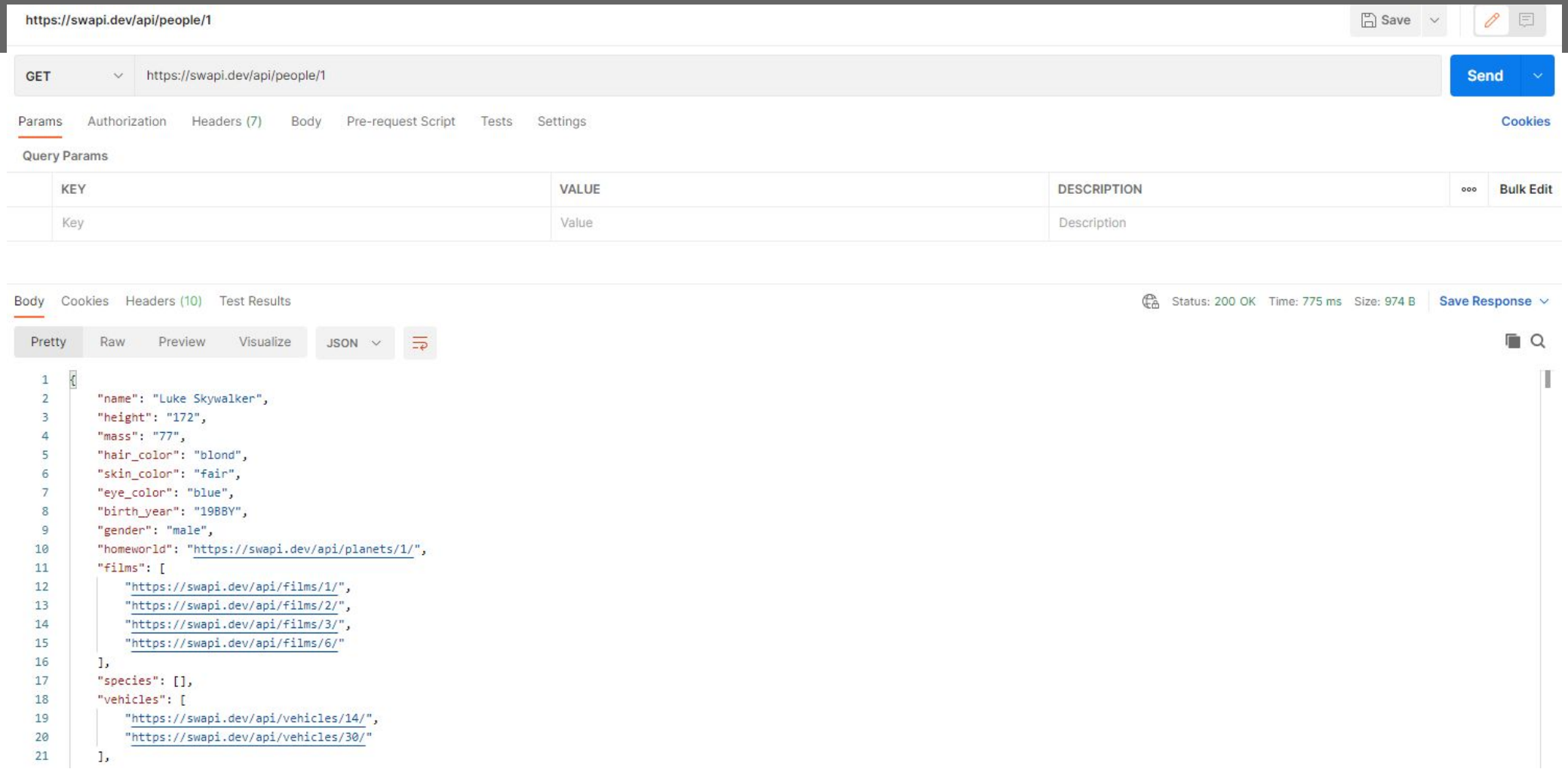
Proporciona una descripción completa y comprensible de los datos en su API, les da a los clientes el poder de pedir exactamente lo que necesitan y nada más, facilita la evolución de las API con el tiempo y habilita poderosas herramientas para desarrolladores.



GraphQL

- Desarrollador por Facebook para uso interno (2015)
- Simplifica la Consulta
- Se apoya sobre HTTP
- Esquema Tipado

GraphQL – Ejemplo REST



https://swapi.dev/api/people/1

Save

GET https://swapi.dev/api/people/1 Send

Params Authorization Headers (7) Body Pre-request Script Tests Settings Cookies

Query Params

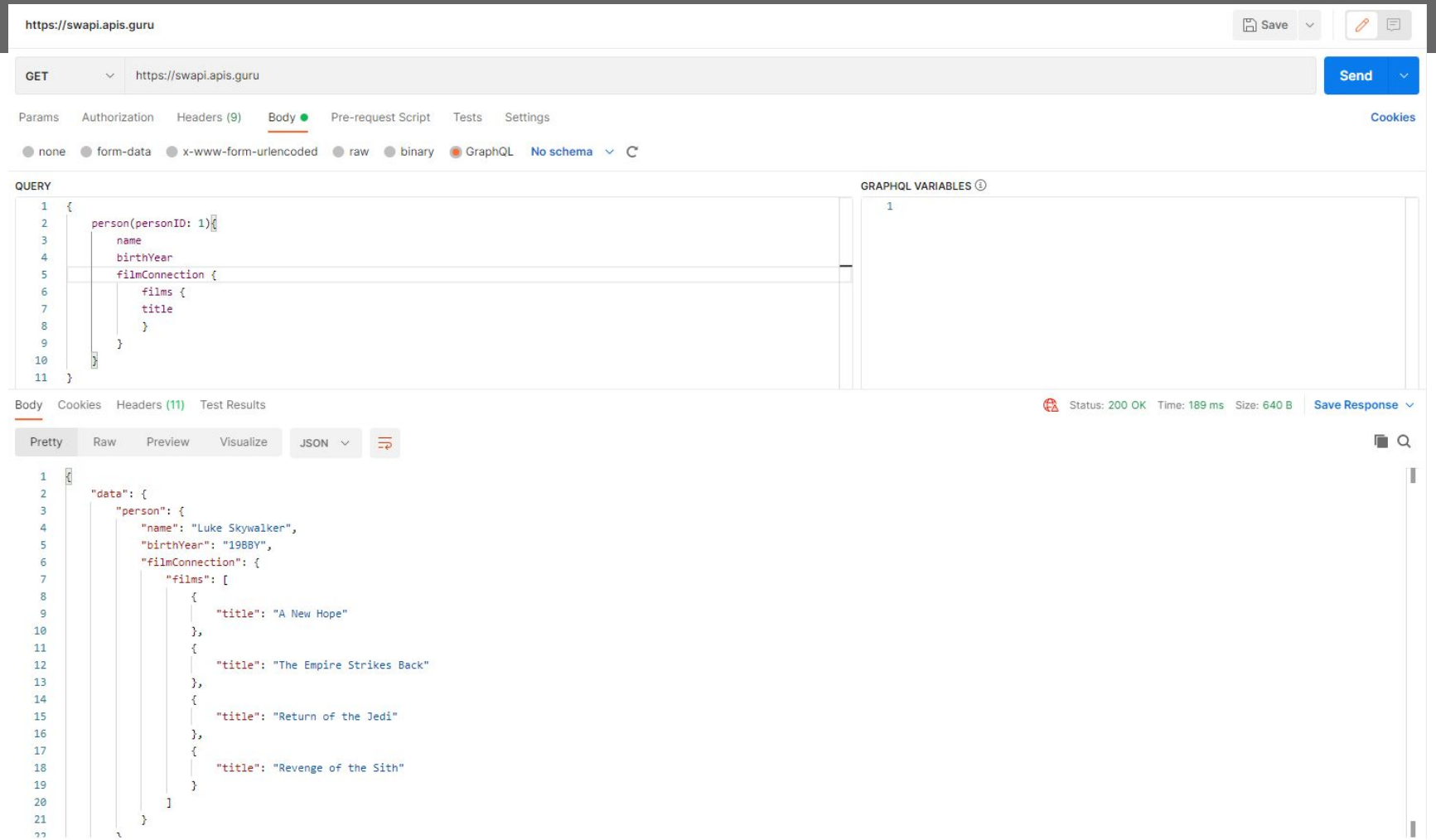
KEY	VALUE	DESCRIPTION	•••	Bulk Edit
Key	Value	Description		

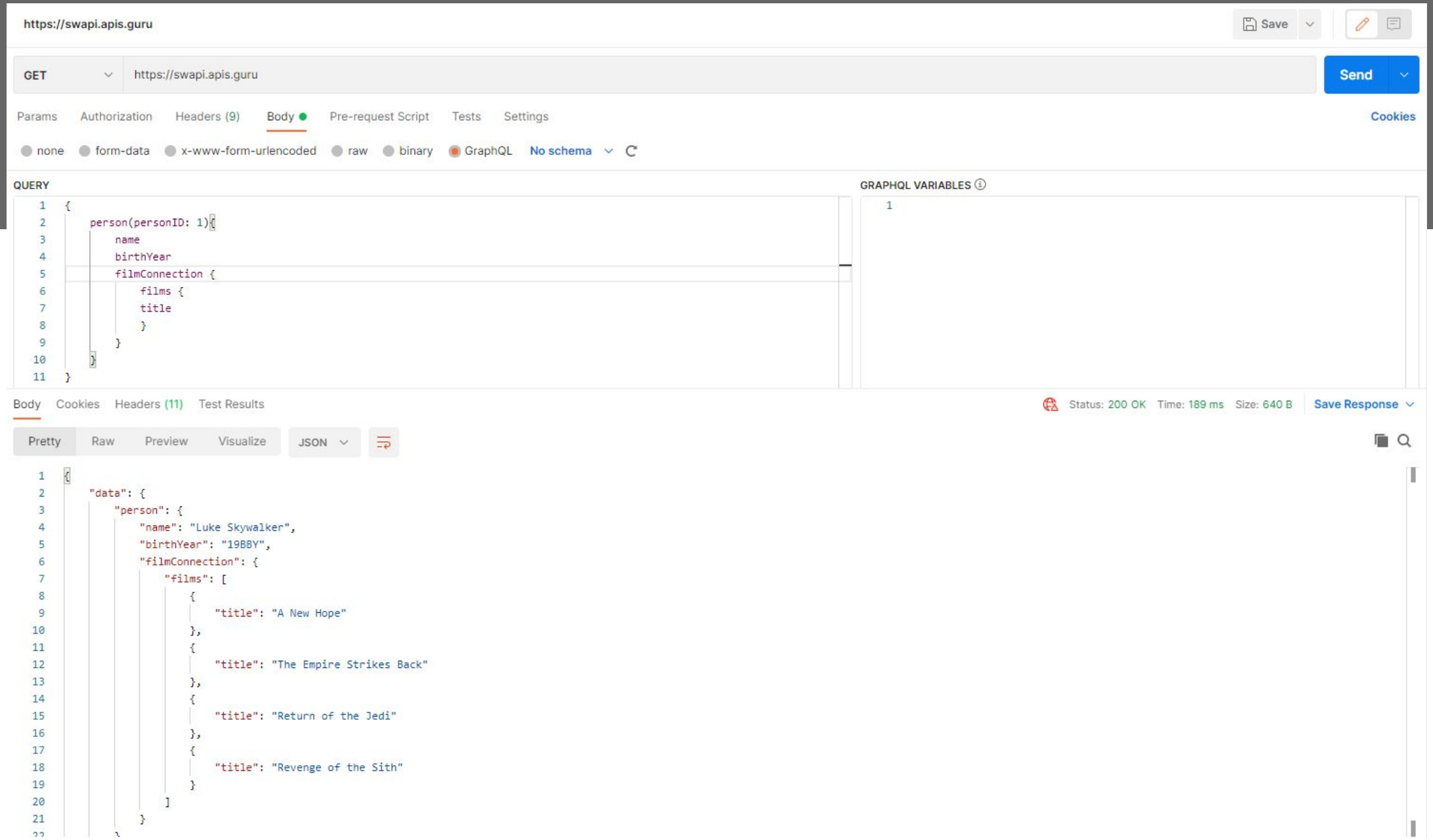
Body Cookies Headers (10) Test Results Status: 200 OK Time: 775 ms Size: 974 B Save Response

Pretty Raw Preview Visualize JSON


```
{
    "name": "Luke Skywalker",
    "height": "172",
    "mass": "77",
    "hair_color": "blond",
    "skin_color": "fair",
    "eye_color": "blue",
    "birth_year": "19BBY",
    "gender": "male",
    "homeworld": "https://swapi.dev/api/planets/1/",
    "films": [
        "https://swapi.dev/api/films/1/",
        "https://swapi.dev/api/films/2/",
        "https://swapi.dev/api/films/3/",
        "https://swapi.dev/api/films/6/"
    ],
    "species": [],
    "vehicles": [
        "https://swapi.dev/api/vehicles/14/",
        "https://swapi.dev/api/vehicles/30/"
    ],
```

GraphQL – Ejemplo Query




https://swapi.apis.guru
Save
GET
https://swapi.apis.guru
Send
Params
Authorization
Headers (9)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
No schema
QUERY
{
person(personID: 1){
name
birthYear
filmConnection {
films {
title
}
}
}
}
GRAPHQL VARIABLES
1
Body
Cookies
Headers (11)
Test Results
Status: 200 OK
Time: 189 ms
Size: 640 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"data": {
"person": {
"name": "Luke Skywalker",
"birthYear": "19BBY",
"filmConnection": {
"films": [
{
"title": "A New Hope"
},
{
"title": "The Empire Strikes Back"
},
{
"title": "Return of the Jedi"
},
{
"title": "Revenge of the Sith"
}
]
}

GraphQL – Ejemplo Query

Query

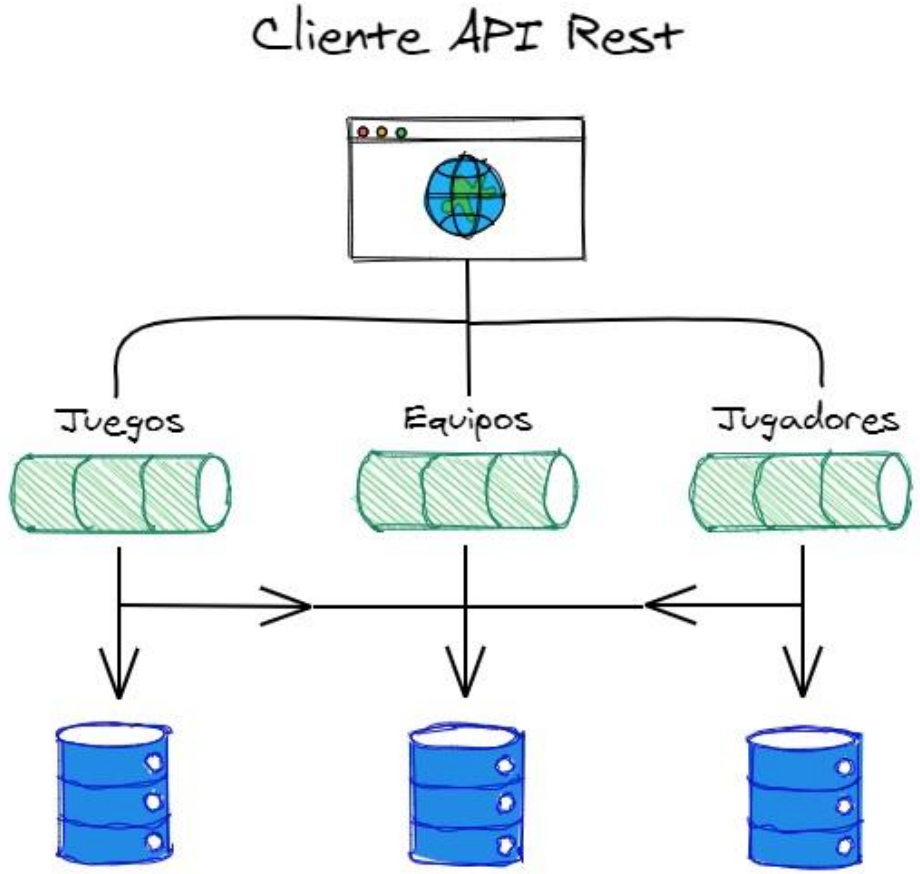
```
{
    person(personID: 1){
        name
        birthYear
        filmConnection {
            films {
             title
            }
        }
    }
}
```

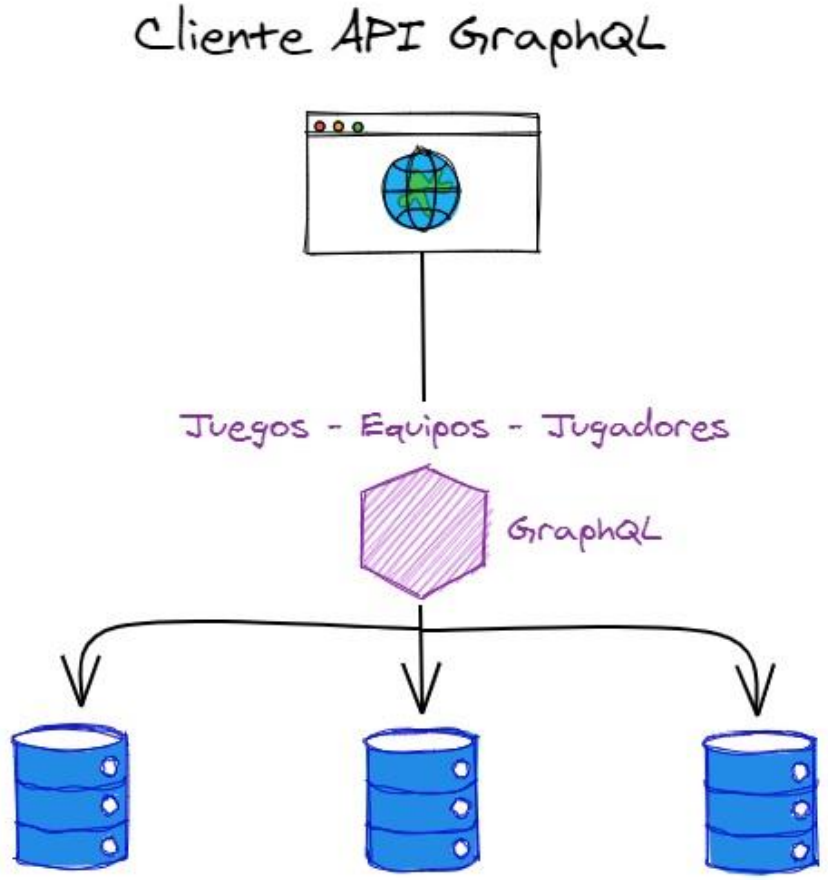
Response


```
{
    "data": {
        "person": {
            "name": "Luke Skywalker",
            "birthYear": "19BBY",
            "filmConnection": {
                "films": [
                    {
                        "title": "A New Hope"
                    },
                    {
                        "title": "The Empire Strikes Back"
                    },
                    {
                        "title": "Return of the Jedi"
                    },
                    {
                        "title": "Revenge of the Sith"
                    }
                ]
            }
        }
    }
}
```

GraphQL – VS REST

GraphQL – VS REST





API - Comparativa

REST	SOAP	GraphQL	gRPC
Sincrónico HTTP	Sincrónico/Asincrónico HTTP, FTP, SMTP	Sincrónico/Asincrónico HTTP, AMQP, MQTT	Sincrónico/Asincrónico HTTP/2
Basado en HTTP (Estados, Métodos, URI)	Basado en intercambio de Mensajes	Basado en intercambio de Mensajes	Basado en intercambio de Mensajes
Múltiples Estándares (Open API, RAML)	Estándar de Especificación de Esquema mediante XML	Estándar de Especificación de Esquema mediante IDL/SDL	Estándar de Especificación de Protobuf
Formato de intercambio JSON	Formato de intercambio XML	Formato de intercambio JSON	Formato de intercambio Serializado (protobuf)

Colas de Mensajes

Una cola de mensajes es una forma de **comunicación asíncrona** de servicio a servicio que se usa generalmente en **arquitecturas de microservicios y sin servidor.**

Los mensajes se **almacenan** en la cola hasta que se **procesan** y eliminan. Cada mensaje se procesa **una sola vez, por un solo consumidor**.

Las colas de mensajes se pueden usar para **desacoplar** procesos pesados, para **acumular trabajo** y para **clasificar cargas de trabajo**.

Colas de Mensajes

Colas de Mensajes

- Proporcionan la comunicación y la coordinación para aplicaciones distribuidas.
- Pueden simplificar de forma significativa la escritura de código para aplicaciones desacopladas
- Mejoran el rendimiento, la fiabilidad y la escalabilidad.
- Permiten a diferentes partes de un sistema comunicarse y procesar las operaciones de forma asíncrona.
- Ofrece un búfer ligero que almacena temporalmente los mensajes.
- Ofrece puntos de enlace que permiten a los componentes de software conectarse a la cola para enviar y recibir mensajes.

Colas de Mensajes

- Los mensajes suelen ser pequeños y pueden ser cosas como solicitudes, respuestas, mensajes de error o, sencillamente, datos.
- Para enviar un mensaje, un componente llamado productor añade un mensaje a la cola.
- El mensaje se almacena en la cola hasta que otro componente, llamado consumidor, lo recupera y hace algo con él.

Colas de Mensajes

- Muchos **productores y consumidores** pueden utilizar la cola, pero **solo un consumidor** procesa **cada mensaje una sola vez** (uno a uno).

- Cuando más de un consumidor debe procesar un mensaje, las colas de mensajes se pueden **combinar implementando el patrón publicación-suscripción** (abanico de salida)

Colas de Mensajes – Algunas implementaciones

- Amazon Simple Queue Service
- Azure Service Bus
- RabbitMQ
- ¿Apache Kafka?
  https://www.cloudamqp.com/blog/when-to-use-rabbitmq-or-apache-kafka.html

  Ejemplo de Uso

  https://www.cloudamqp.com/blog/softonic-userstory-rabbitmq-eventbased-communication.html

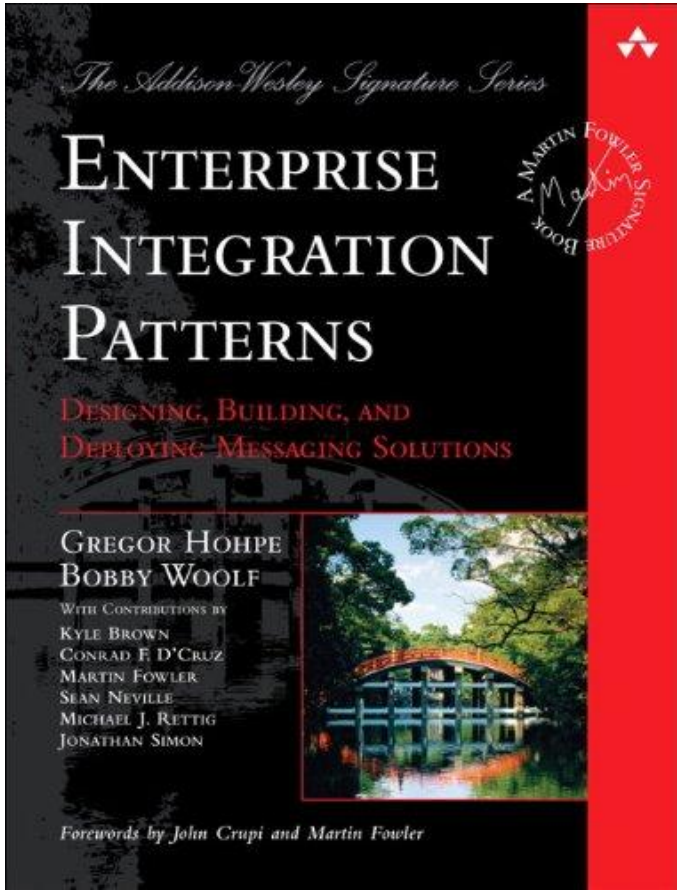
Base de Datos Compartida

- Se utiliza una **Base de Datos** central para **integrar** los **componentes**.
- Está centrado en la **integración de datos**, no de uso de funcionalidad.
- Permite integración **asincrónica y sincrónica**.
- Es **simple** de implementar.
- Cada componente puede estar resuelto en **diferente stack tecnológico**.
- Cada **componente de lógica de negocio** es **independiente** en su **disponibilidad**.

Base de Datos Compartida

- Puede generar problemas de performance.
- El acoplamiento a los modelos puede ser elevada.
- Es difícil de cambiar.
- Se debe considerar la seguridad (CIA) de los datos.

Para Seguir: Bibliografía



Aplicaciones en la Nube vs Tradicionales

Aplicativos en la nube - Desafíos

- Cambia la forma en que se diseñan las aplicaciones.
- En lugar de ser "monolitos", las aplicaciones se descomponen en servicios menores y descentralizados.
- Los servicios se comunican a través de API o mediante el uso de eventos o de mensajería asincrónica.
- Las aplicaciones se escalan horizontalmente, agregando nuevas instancias, tal y como exigen las necesidades.

Aplicaciones en la Nube vs Tradicionales

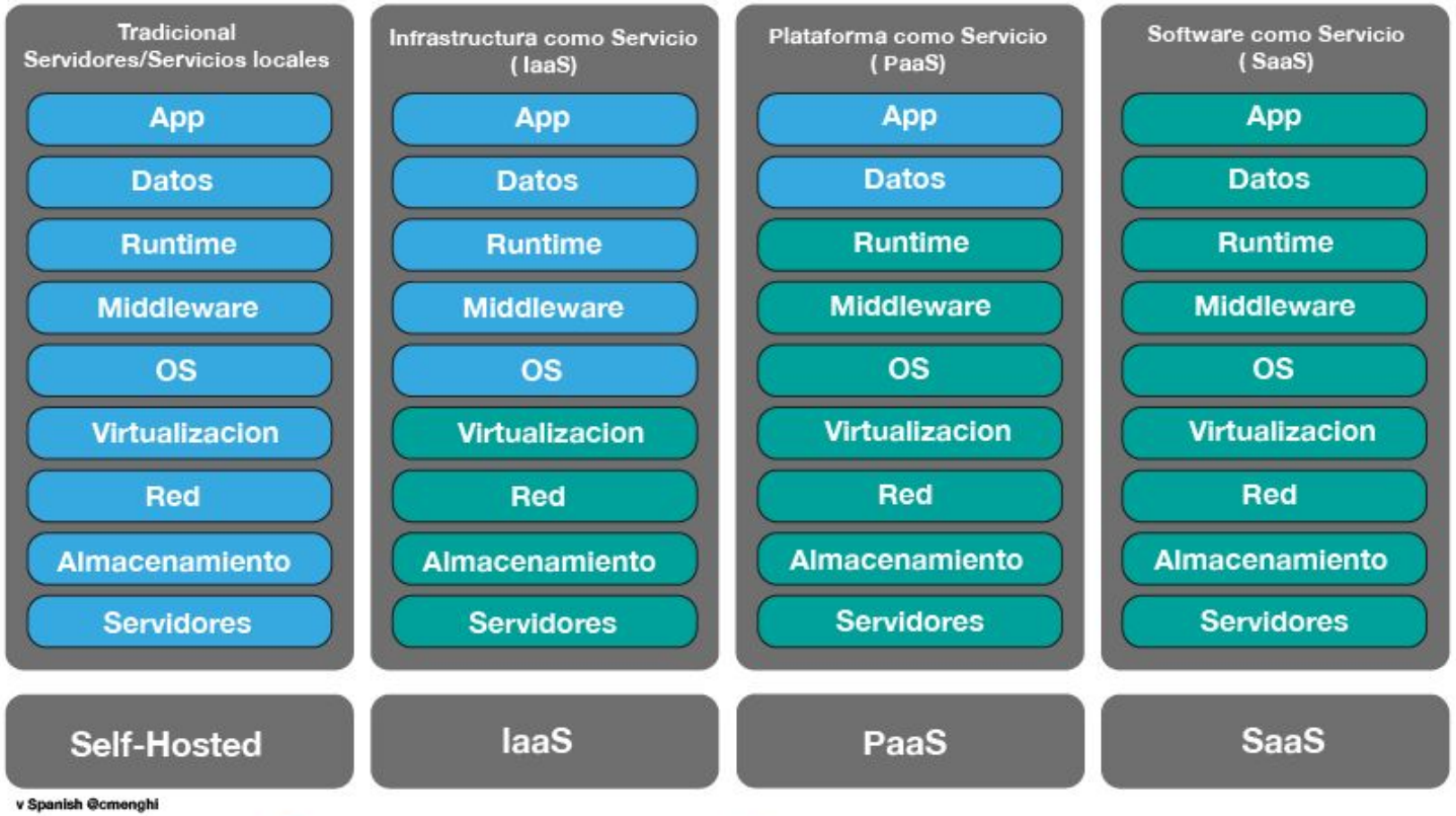
Aplicativos en la nube - Desafíos

- El estado de las aplicaciones se distribuye.
- Las operaciones se realizan en paralelo y de forma asincrónica.
- Las aplicaciones deben ser resistentes cuando se produzcan errores.
- Las implementaciones deben estar automatizadas y ser predecibles.
- La supervisión y la telemetría son fundamentales para obtener una visión general del sistema.

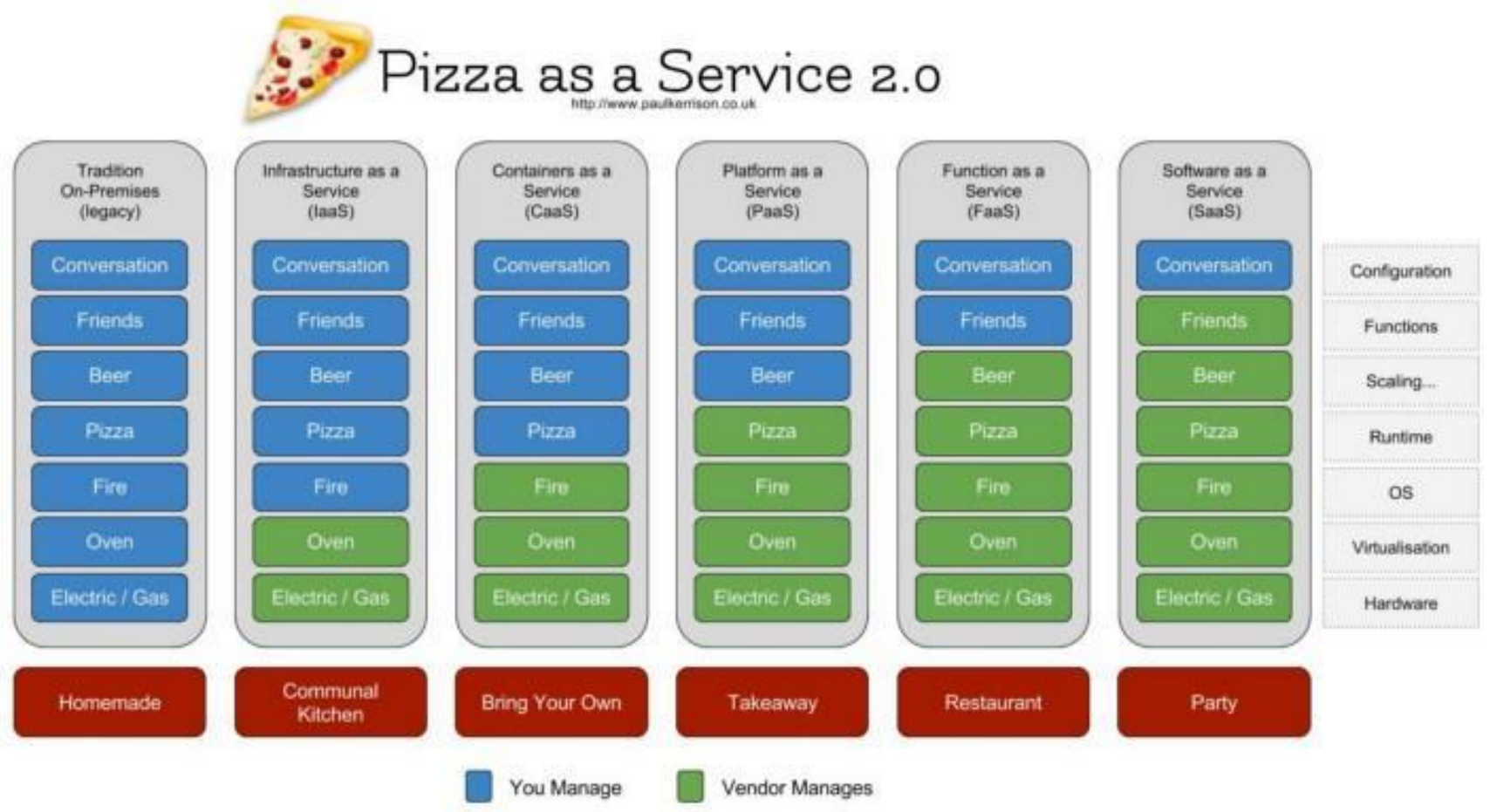
Aplicaciones en la Nube vs Tradicionales

Local – Tradicional	Nube – "Moderna"
• Monolítica • Diseñada para una escalabilidad predecible • Base de datos relacional • Procesamiento sincronizado • Diseño para evitar errores • Actualizaciones grandes ocasionales • Administración manual • Servidores de copo de nieve	• Descompuesto • Diseñado para un escalado elástico • Persistencia Poliglota (combinación de tecnologías de almacenamiento) • Procesamiento asincrónico • Diseño para errores • Pequeñas actualizaciones frecuentes • Administración automatizada • Infraestructura inmutable

Un poco de Infraestructura



Un poco de Infraestructura



Componentes: IAM / IDM

- Es un componente que gestiona las identidades y los accesos.

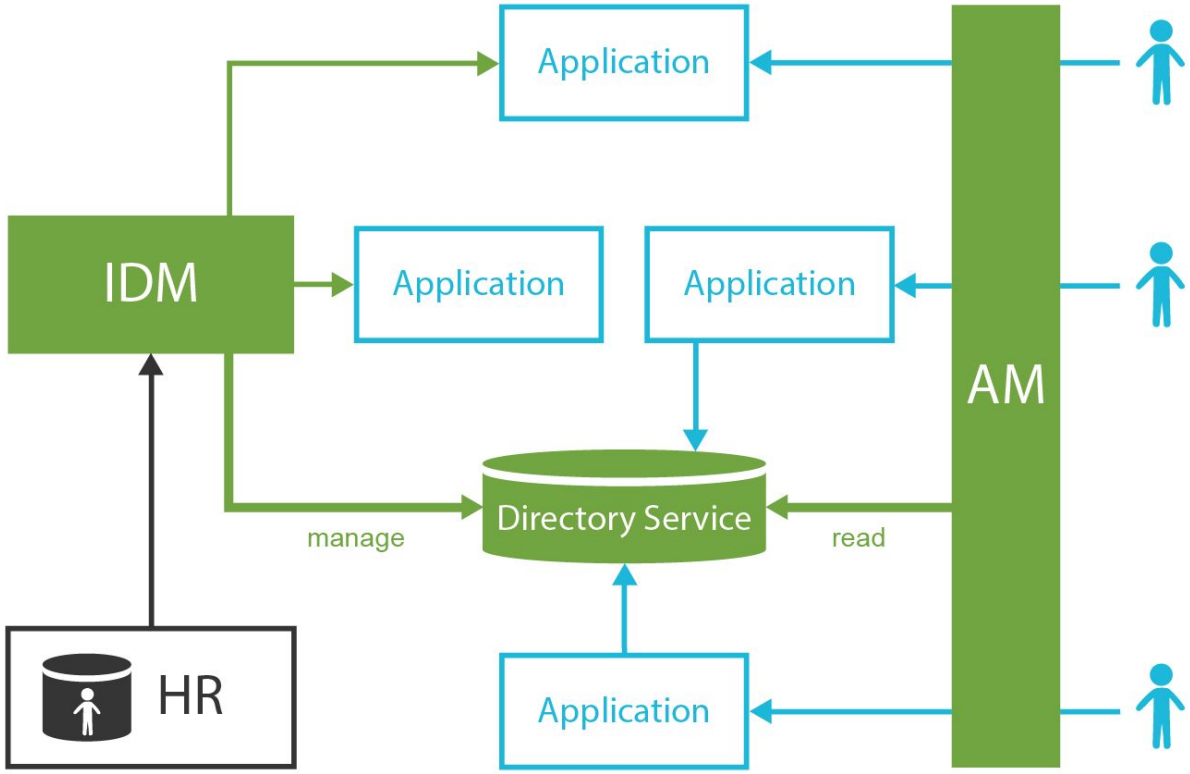
- IAM: Acceso
  - Define Estrategias para el acceso a través de varias tecnologías, como contraseñas, biometría, autenticación multifactor y otras identidades digitales

- IDM: Identidad
  - Se centra más en la identidad de un usuario (o nombre de usuario), sus roles y permisos, y los grupos a los que pertenece

Autenticación vs Autorización

- La autenticación confirma que los usuarios son quienes dicen ser, validando la identidad del usuario.
- La autorización otorga a los usuarios autenticados permiso para obtener acceso a un recurso.

Autenticación vs Autorización



IAM / IDM

- Keycloak
- OpenIAM
- Soffid
- Auth0

¿Qué es un SSO?

Es un componente que permite tener acceso a múltiples aplicaciones ingresando solo con una cuenta a los diferentes sistemas y recursos.

Es posible acceder mediante una única "contraseña" y se desea evitar el ingreso repetitivo de estas cada vez que el usuario se desconecte del servicio.

Para los usuarios supone una gran comodidad ya que identificándose solo una vez es posible mantener la sesión válida para el resto de las aplicaciones que hacen uso del SSO.

SSO - Características

- Acelera el acceso de los usuarios a sus aplicaciones
- Reduce la carga de memorizar diversas contraseñas
- Fácil de implementar y conectar a nuevas fuentes de datos
- Al fallar SSO se pierde acceso a todos los sistemas relacionados
- Suplantación de identidades en los accesos externos de los usuarios

Componentes: CDN

- Una red de entrega de contenido (CDN) está formada por un grupo de servidores distribuidos geográficamente que trabajan juntos para ofrecer una entrega rápida de contenido de Internet.

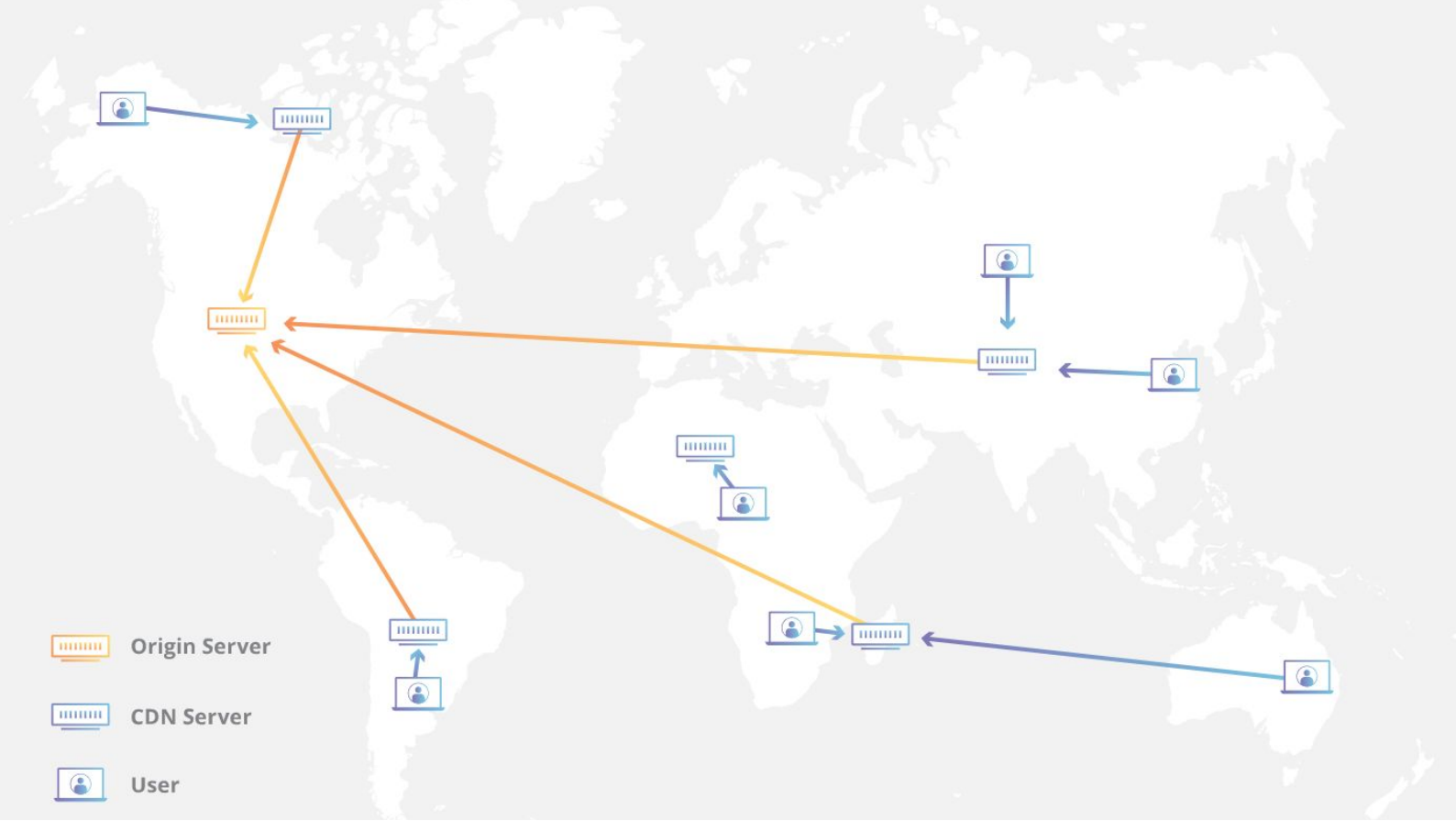
- Permite la transferencia rápida de activos necesarios para cargar contenido de Internet, incluidas páginas HTML, archivos js, css, imágenes y vídeos.

- Puede ayudar a proteger sitios web contra algunos ataques maliciosos comunes, como los ataques de denegación de servicio distribuido (DDOS).

Componentes: CDN Ventajas

- Mejora de los tiempos de carga del sitio web
- Reducción de los costos de ancho de banda
- Aumento de la disponibilidad de contenido y la redundancia
- Mayor seguridad del sitio web

Componentes: CDN Arquitectura



Componentes: Cache

La caché HTTP almacena una respuesta asociada con una solicitud y reutiliza la respuesta almacenada para solicitudes posteriores.

Hay varias ventajas de reusabilidad. En primer lugar, dado que no es necesario enviar la solicitud al servidor origen, cuanto más cerca están el cliente y la caché, más rápida será la respuesta. El ejemplo más típico es cuando el navegador almacena una caché para las solicitudes.

El servidor de origen no necesita procesar la solicitud, por lo que no necesita parsear y enrutar la solicitud, restaurar la sesión en función de la cookie, consultar los resultados de la base de datos o renderizar la plantilla. Eso reduce la carga en el servidor.

El funcionamiento adecuado de la memoria caché es fundamental para la salud del sistema.

Componentes: Características Cache

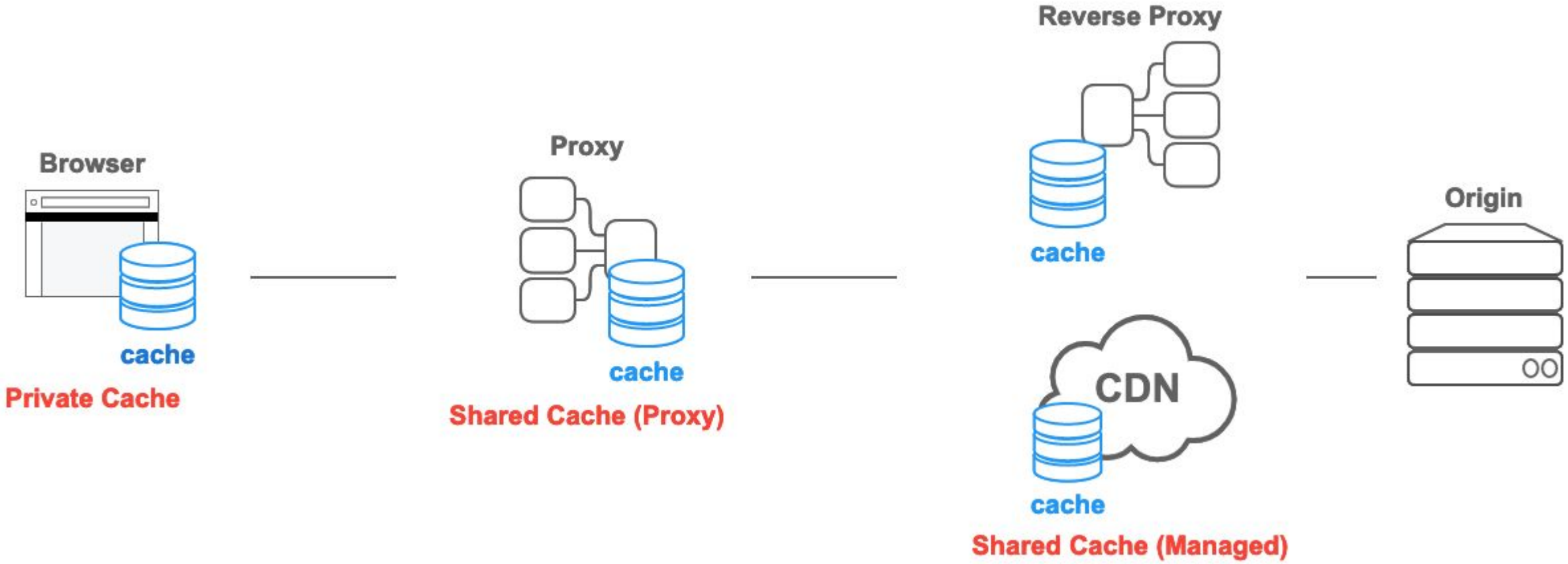
- Privada (Usuario)
- Compartida
  - Proxy Reverso
  - CDN
- Tiempo de Expiración

Componentes: Ejemplo Cache

URL	Response
https://example.com/index.html	<!doctype html>...
https://example.com/style.css	body { ...
https://example.com/script.js	function main () { ...

URL	Accept-Language	Response
https://example.com/index.html	ja-JP	<!doctype html>...
https://example.com/index.html	en-US	<!doctype html>...
https://example.com/style.css	ja-JP	body { ...
https://example.com/script.js	ja-JP	function main () { ...

Componentes: Arquitectura Cache



Componentes: Cache

¿Qué otras Cache podemos implementar?

Componentes: Cache

¿Qué otras Cache podemos implementar?

- Hibernate Caching (Cache en la capa de Persistencia)
- ….

Gracias