



# Arquitectura de sistemas, comportamiento y optimización

Week 9

M.Sc. Juan Luis Flores Pineda

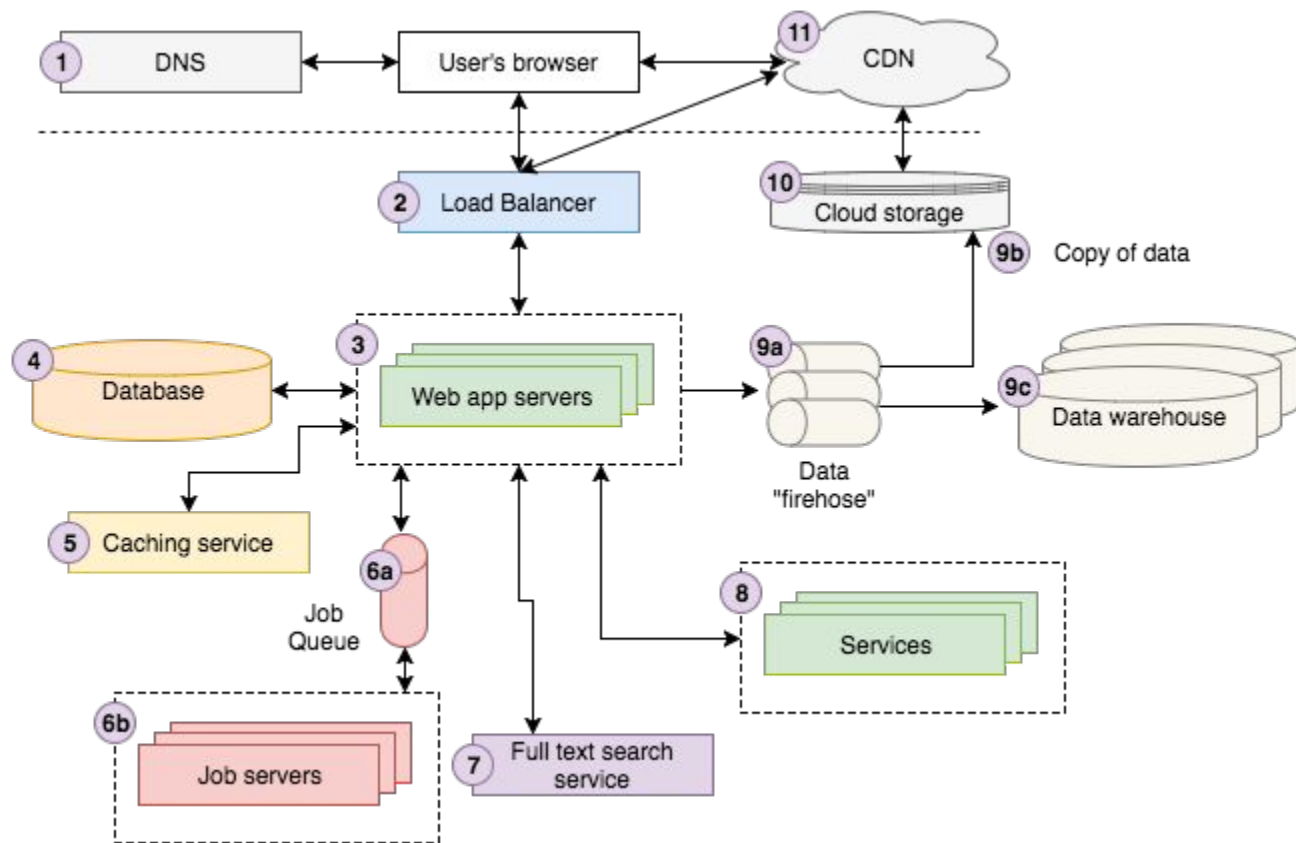
# Agenda

- Arquitectura Web
  - ◆ Introducción
  - ◆ Conceptos básicos
  - ◆ Servicios REST
  - ◆ Application Programming Interface (API)
- Modelo y Diseño de Arquitectura - Móviles
  - ◆ Aplicaciones Híbridas
  - ◆ Cross Compiled / Cross Platform
- Modelo y Diseño de Arquitectura - Web
  - ◆ RIA
  - ◆ SPA (Single Page Applications)



# Arquitectura Web

## Introducción



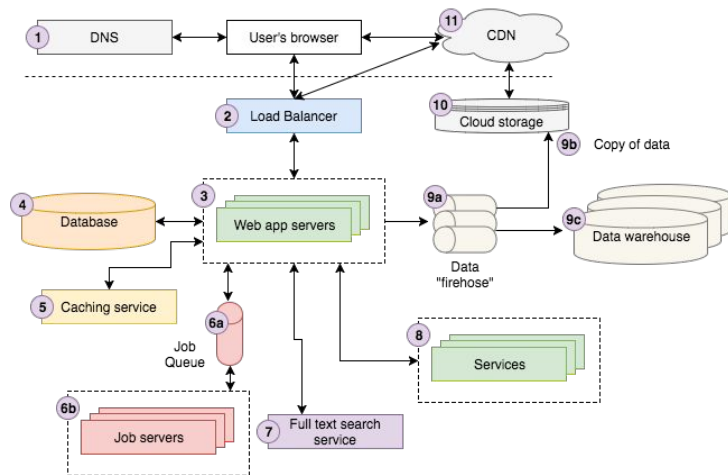


# Conceptos Básicos

# 1 - DNS - Conceptos Básicos

Significa "Domain Name System" y es una tecnología troncal que hace posible la red mundial. En el nivel más básico, el DNS proporciona una búsqueda de clave / valor de un nombre de dominio a una dirección IP.

Por ejemplo, google.com a una dirección IP 74.125.224.72



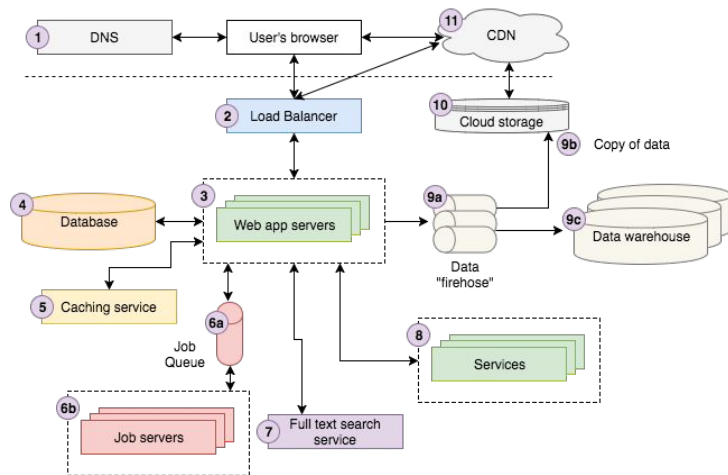
## 2 - Load Balancer - Conceptos Básicos

Existen dos tipos de escalado de recursos :

- Horizontal: se agregan más máquinas a su grupo de recursos.
- Vertical: Se agrega más potencia a una máquina existente (CPU, RAM, Disco, etc).

Normalmente en una arquitectura web se tiende a escalar de forma horizontal debido a:

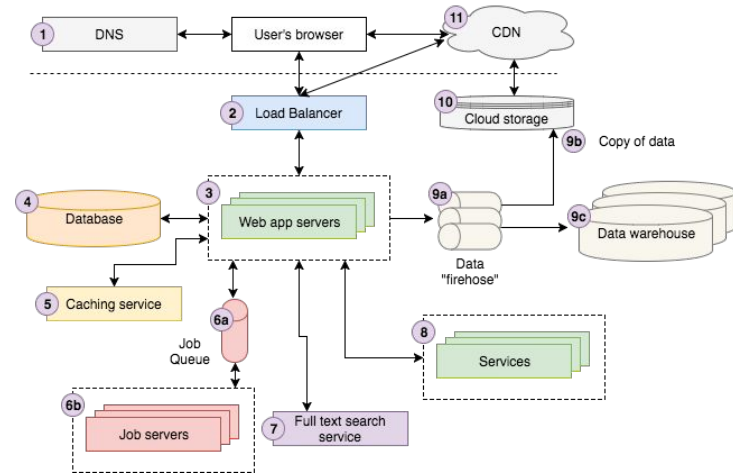
- El Escalado Vertical tiene un límite.
- Degradación de la red.
- Permite planificar interrupciones y que la aplicación continúe en la nube.
- Vuelve a la aplicación tolerante a fallas.



## 2 - Load Balancer - Conceptos Básicos.

Load Balancer o balanceador de carga permite equilibrar la carga entre los diferentes servidores (escalado Horizontal), de manera de tener un buen rendimiento.

Permiten escalar de forma horizontal, debido a que se encargan de enrutar las solicitudes entrantes a uno de los servidores de la aplicación y busca distribuir las solicitudes de manera eficiente para que ninguno de los servidores esté sobrecargado.



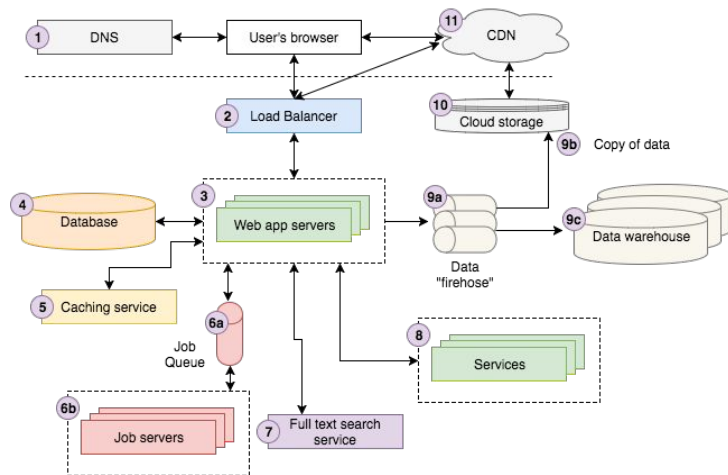


# 3- Servidores de aplicaciones Web -Conceptos Básicos

Ejecutan la lógica comercial central que maneja la solicitud de un usuario y envía de vuelta HTML al navegador del usuario.

Estas generalmente se comunican con una variedad de infraestructura como: bases de datos, capas de almacenamiento en caché, colas de trabajos, servicios de búsqueda, microservicios, data-login, etc.

Las implementaciones del servidor de aplicaciones requieren elegir un lenguaje de programación específico normalmente.

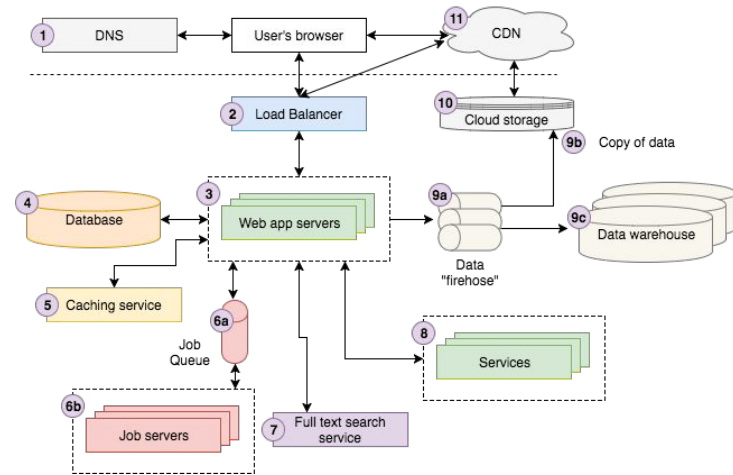


# 4 - Servidores de DB - Conceptos Básicos

Las bases de datos proporcionan formas de definir sus estructuras de datos, insertar datos nuevos, encontrar datos existentes, actualizar o eliminar datos existentes, realizar cálculos a través de los datos, etc.

En la mayoría de los casos, los servidores de aplicaciones web se comunican directamente con un servidor de base de datos.

También, cada servicio puede tener su propia base de datos aislada del resto de la aplicación.

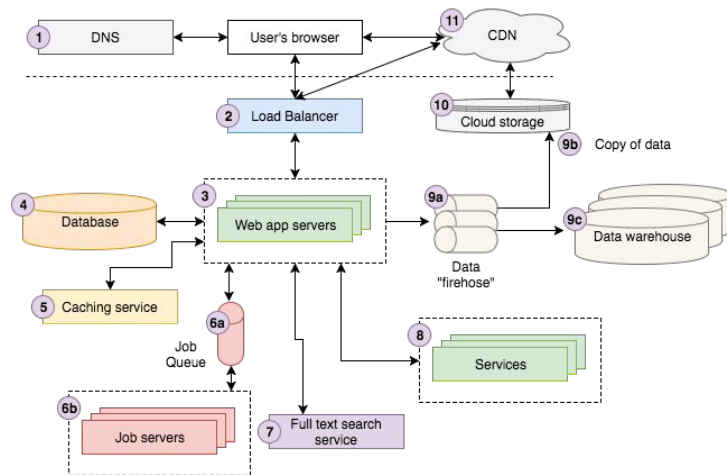


# 5 - Servicio de Caché - Conceptos Básicos.

Un servicio de almacenamiento en caché proporciona un almacén de datos clave, permitiendo guardar y buscar información en un tiempo menor.

Por lo general, las aplicaciones aprovechan los servicios de almacenamiento en caché para guardar los resultados de cálculos costosos para que sea posible recuperar los resultados de la memoria caché en lugar de volver a calcularlos la próxima vez que se necesiten.

Una aplicación puede almacenar en caché los resultados de una consulta de base de datos, llamadas a servicios externos, HTML para una URL determinada y muchos más.

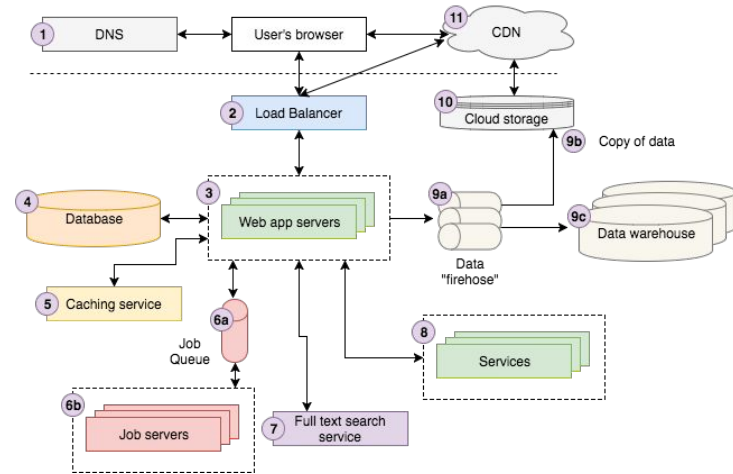


# 6 - Colas de Trabajo y Servidores -Conceptos Básicos.

La mayoría de las aplicaciones web necesitan hacer un trabajo asíncrono detrás de escena que no está directamente asociado con la respuesta a la solicitud de un usuario.

Si bien existen diferentes arquitecturas que permiten realizar trabajos asincrónicos, la más ubicua es la arquitectura de cola de trabajos que consta de dos componentes:

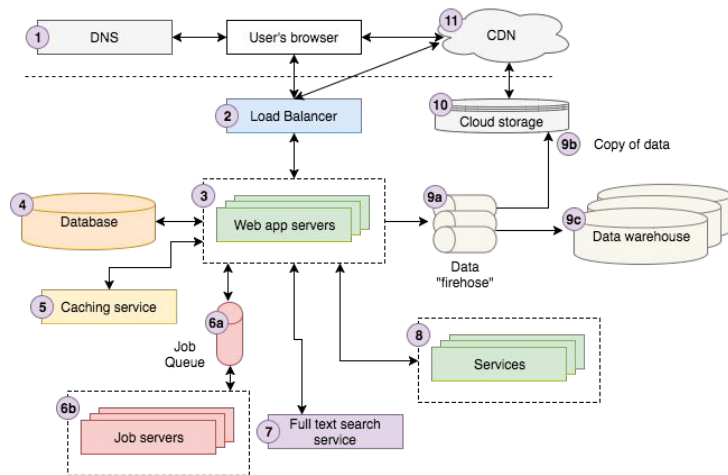
- una cola de "trabajos" que deben ejecutarse.
- Uno o más servidores de trabajo que ejecutan los trabajos en la cola.



# 7 - Full-Text Search Service - Conceptos Básicos.

La mayoría de las aplicaciones web admiten algún tipo de función de búsqueda en la que un usuario proporciona una entrada de texto y la aplicación devuelve los resultados más relevantes.

La tecnología que impulsa esta funcionalidad generalmente se conoce como "Full-Text", que aprovecha un índice invertido para buscar rápidamente documentos que contienen las palabras clave de consulta.



## 7 - Full-Text Search Service - Conceptos Básicos.

Documents (Photo titles)	
id	title
1	Man running in the mountains
2	Mountains with snow
3	Man running marathon



Inverted Index	
keyword	photo_ids
man	1, 3
running	1, 3
mountains	1, 2
snow	2
marathon	3

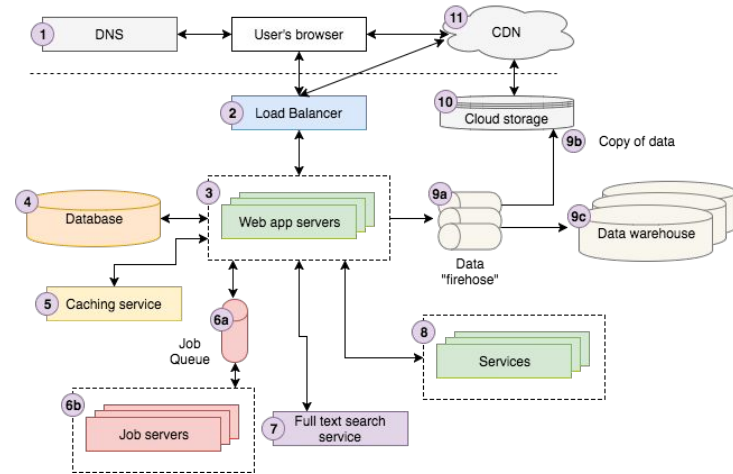
Lo común es utilizar un servicio de búsqueda separado que computa, almacena el índice invertido y proporciona una interfaz de consulta como: ElasticSearch (amazon), Apache Solr, Algolia y otros.

# 8 - Servicios - Conceptos Básicos.

Una vez que una aplicación alcanza cierta escala, es probable que haya ciertos "servicios" diseñados para ejecutarse como aplicaciones separadas.

Posiblemente estos no están expuestos externamente pero la aplicación interactúa con ellos.

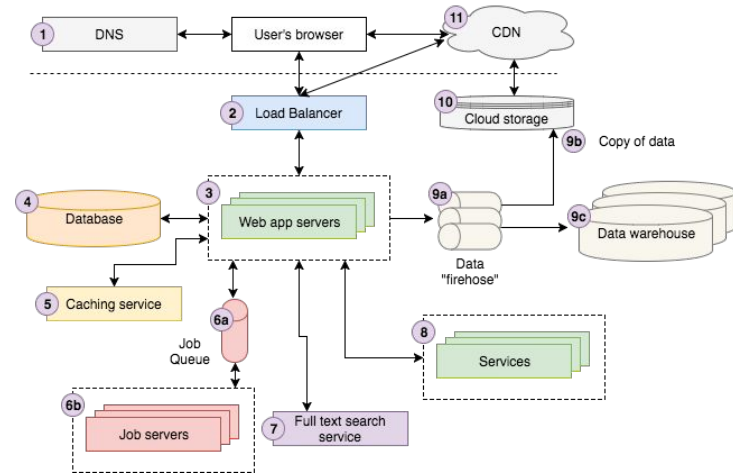
Ejemplo de estos son: servicios de autenticación de usuarios, control de usuarios(Cognito AWS), contenido (assets), servicios de pago, conversiones de datos, etc.



# 9 - Datos - Conceptos Básicos.

Casi todas las aplicaciones en estos días, una vez que alcanza una cierta escala, aprovechan una tubería de datos para garantizar que los datos se puedan recopilar, almacenar y analizar. Una tubería típica tiene tres etapas principales:

- A. Firehose: La aplicación envía datos, sobre las interacciones del usuario y proporciona una interfaz para procesar los datos.
- B. Warehouse: permiten hacer análisis e informes sobre datos estructurados y semiestructurados de diferentes fuentes.
- C. Almacenamiento de datos transformados para su uso en el futuro.



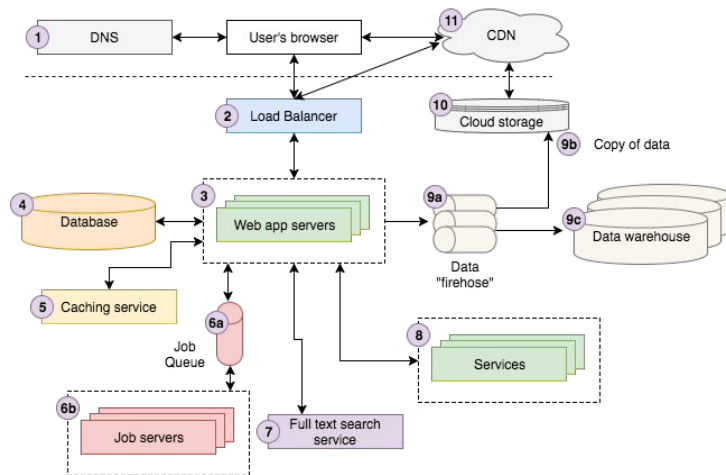


# 10 - Almacenamiento en la nube - Conceptos Básicos.

Es una forma simple y escalable de almacenar, acceder y compartir datos a través de Internet.

Puede usarlo para almacenar y acceder a más o menos cualquier cosa que almacene en un sistema de archivos local con los beneficios de poder interactuar con él a través de una API RESTful a través de HTTP.

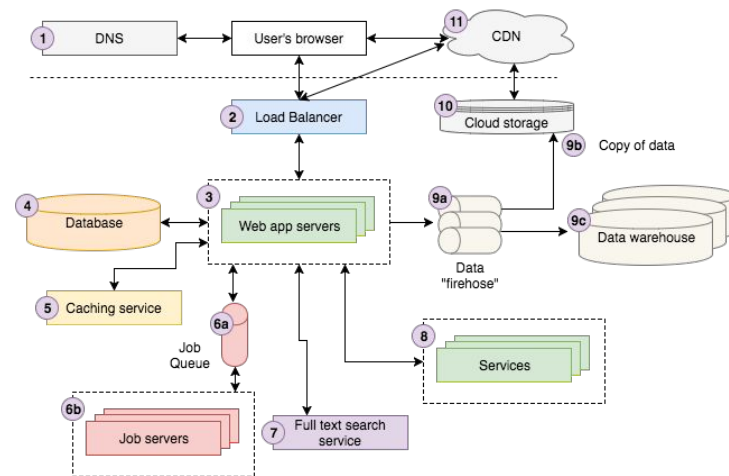
Amazon S3 es el almacenamiento en la nube más popular disponible en la actualidad.



# 11 - CDN Conceptos Básicos.

Significa "Content Delivery Network" y la tecnología proporciona una forma de servir activos como HTML estático, CSS, Javascript e imágenes en la web mucho más rápido que servirlos desde un único servidor de origen.

Funciona mediante la distribución del contenido en muchos servidores "perimetrales" de todo el mundo para que los usuarios terminen descargando activos de los servidores "perimetrales" en lugar del servidor de origen.





# Arquitectura Web

## Servicios REST



¿Qué es REST?

# REpresentational State Transfer

Es un estilo arquitectónico, permite establecer estándares informáticos en la web, facilitando la comunicación entre sistemas.

Los sistemas RESTful, se caracterizan por manejar transacciones independientes y establecer la separación entre el cliente y servidor.

# Cliente y Servidor

Debido a la naturaleza de la arquitectura la implementación del cliente y el servidor se pueden realizar de forma independiente. El código se puede cambiar en cualquiera de los dos sin afectar el funcionamiento del otro.

Todo esto se logra mientras cada uno de los lados sepa que formato de mensajes debe enviar al otro.

Esta arquitectura permite mejorar la interfaz entre plataformas y la escalabilidad, debido a que cada uno maneja sus tareas de forma independiente.

Cada componente puede evolucionar de forma independiente.

# Comunicación Cliente

REST requiere que un cliente realice una solicitud al servidor para recuperar o modificar datos en el servidor.

Un Request generalmente consiste en:

- A. Verbo HTTP, que define qué tipo de operación realizar.
- B. Header, que permite al cliente transmitir información sobre la solicitud.
- C. Path del recurso.
- D. Cuerpo de mensaje (opcional) que contiene datos.

# A. Verbos HTTP

Existen 4 verbos básicos HTTP, que son utilizados en los request para interactuar con los recursos en un sistema REST:

- GET: permite recuperar un recurso específico (por id) o una colección de recursos.
- POST: permite crear un nuevo recurso
- PUT: Actualizar un recurso específico por ID.
- DELETE: eliminar un recurso específico por ID.



## B. Headers y Parámetros

En el header de la solicitud, el cliente envía el tipo de contenido que puede recibir del servidor. Esto se define en el *accept* y garantiza que el servidor, no envíe datos que el cliente no pueda comprender o procesar.

Las opciones para los tipos de contenido son MIME Types (Extensiones multipropósito de correo de Internet).

Los tipos MIME, utilizados para especificar los tipos de contenido en el campo Accept, consisten en un tipo y un subtipo. Están separados por una barra inclinada (/).

**Nota:** *accept* puede especificar más de un tipo de formato.

## B. Headers y Parámetros

Los más comunes son:

- text/html: indica que es un archivo de texto que contiene HTML.
- text/css: indica que es un archivo de texto contiene CSS.
- text/plain: indica que es un archivo de texto genérico o plano.
- Formatos de imagen: **JPEG 2000, JPEG XR, WebP** ,PNG, JPEG, GIF
- Formatos de audio: audio/wav, audio/mpeg
- Formatos de video: video/mp4, video/ogg
- Formatos de application: application/json, application/pdf, application/xml, application/octet-stream, application/xhtml.

**Nota:** si un cliente espera un tipo de formato y recibe otro no podrá reconocer el contenido.

## C. Path del Recurso

Es importante que los request contengan una ruta al recurso al cual quieren realizar la operación.

En las API RESTful las rutas deben diseñarse de manera de ayudar al cliente a conocer qué está sucediendo. Por estándar:

- La primera parte de la ruta debe ser la forma plural del recurso.
- Las rutas deben ser jerárquicas y descriptivas.
- Las rutas deben contener la información necesaria para ubicar un recurso con el grado de especificidad necesario.
- Cuando se hace referencia a una lista o colección de recursos no hace falta agregar un ID, eso solo se agrega si necesitamos identificar una ruta en específico.

## D. Cuerpo del mensaje

Este es un field opcional, normalmente es utilizado cuando se crean nuevos recursos ejemplo de llamada utilizando POST:

Request:

```
POST http://www.myrestaurant.com/dishes/
```

Body -

```
{
  "dish": {
    "name": "Avocado Toast",
    "price": 8
  }
}
```

# Comunicación Servidor

Response es una respuesta a un request, es una respuesta que da un servidor.

En los response tenemos dos partes importantes, adicionales:

- Tipo de contenido
- Códigos de respuestas



# Tipo de Contenido

En los casos en que el servidor envía una carga de datos al cliente, el servidor debe incluir *content-type* en el encabezado de la respuesta. Este campo de encabezado de *content-type* alerta al cliente sobre el tipo de datos que está enviando en el cuerpo de respuesta.

Estos tipos de contenido son tipos MIME, tal como se encuentran en el campo *accept* del encabezado de la solicitud.

**Nota:** El tipo de contenido que el servidor devuelve en la respuesta debe ser una de las opciones que el cliente especificó en el campo aceptar de la solicitud.

# Códigos de respuesta

Los responses del servidor contienen códigos de estado para alertar al cliente sobre información acerca del éxito de la operación. No necesitamos conocer todos los códigos de estado, solo los más comunes y cómo se usan:

200 (OK)	Esta es la respuesta estándar para solicitudes HTTP exitosas.
201 (CREATED)	Esta es la respuesta estándar para una solicitud HTTP que resultó en la creación exitosa de un elemento.
204 (NO CONTENT)	Esta es la respuesta estándar para solicitudes HTTP exitosas, donde no se devuelve nada en el cuerpo de la respuesta.
400 (BAD REQUEST)	La solicitud no se puede procesar debido a una sintaxis de solicitud incorrecta, un tamaño excesivo u otro error del cliente.
403 (FORBIDDEN)	El cliente no tiene permiso para acceder a este recurso.
404 (NOT FOUND)	El recurso no se pudo encontrar en este momento. Es posible que se haya eliminado o que aún no exista.
500 (INTERNAL SERVER ERROR)	La respuesta genérica para una falla inesperada si no hay más información específica disponible.

# Restful

Tal como se define en la tesis de Roy Fielding, “Architectural Styles and the Design of Network-based Software Architectures”, las API son RESTful siempre que cumplan con las 6 limitaciones principales de un sistema RESTful:

1. **Arquitectura cliente-servidor:** la arquitectura REST está compuesta por clientes, servidores y recursos, y administra las solicitudes con HTTP.
2. **Sin estado:** el contenido de los clientes no se almacena en el servidor entre las solicitudes, sino que la información sobre el estado de la sesión se queda en el cliente. En su lugar, la información sobre el estado de la sesión está en posesión del cliente.



# Restful

3. **Capacidad de caché:** el almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente-servidor.
4. **Sistema en capas:** las interacciones cliente-servidor pueden estar mediadas por capas adicionales, que pueden ofrecer otras funciones, como el equilibrio de carga, los cachés compartidos o la seguridad.
5. **Código de demanda (opcional):** los servidores pueden extender las funciones de un cliente transfiriendo código ejecutable.

# RESTful - Interfaz Uniforme

6. **Interfaz uniforme:** esta limitación es fundamental para el diseño de las API de RESTful e incluye 4 aspectos:
  1. Identificación de recursos en las solicitudes: los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.
  2. Administración de recursos mediante representaciones: los clientes reciben archivos que representan los recursos. Estas representaciones deben tener la información suficiente como para poder ser modificadas o eliminadas.

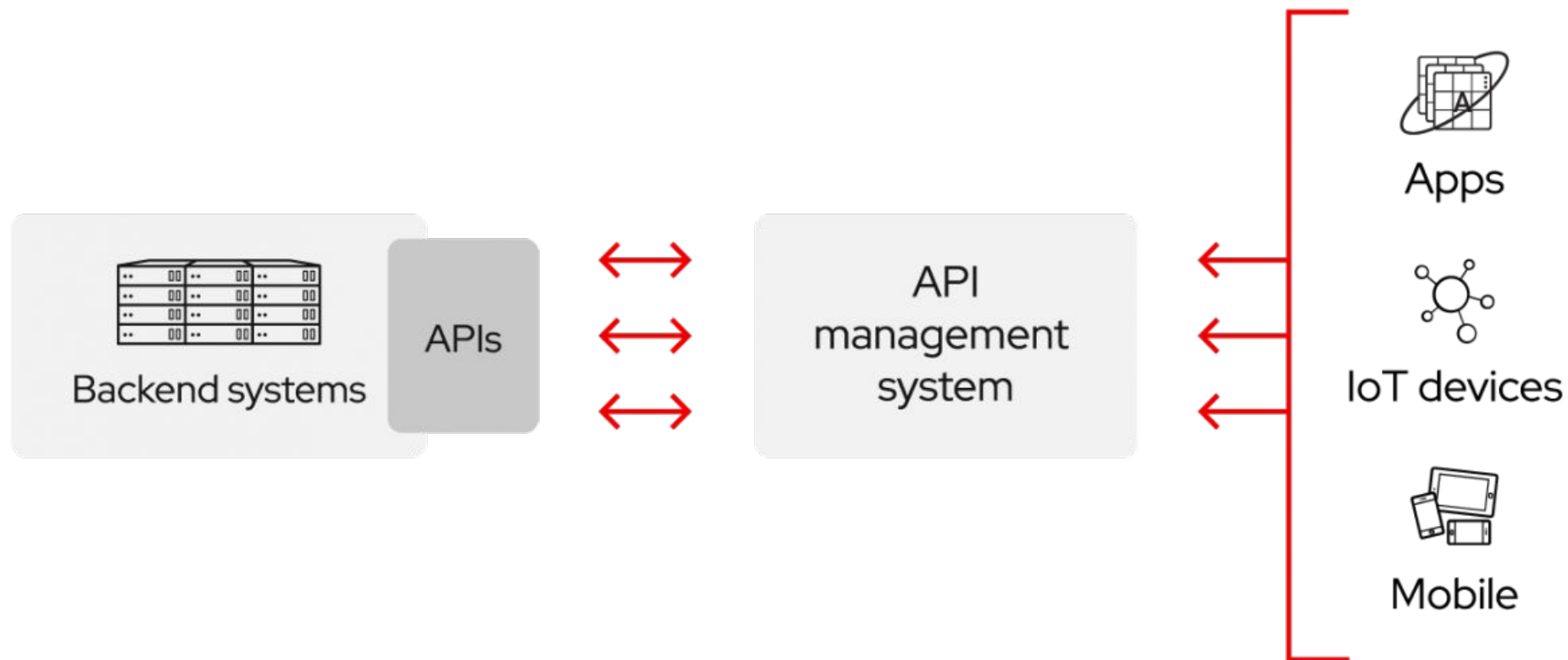
# RESTful - Interfaz Uniforme

3. Mensajes autodescriptivos: cada mensaje que se devuelve al cliente contiene la información suficiente para describir cómo debe procesar la información.
4. Hipermedios es el motor del estado de la aplicación: después de acceder a un recurso, el cliente REST debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están disponibles actualmente.



# Arquitectura Web

**Application Programming Interface (API)**





# Breve Historia

Las API surgieron los primeros días de la informática, mucho antes que la computadora personal.

En esa época, una API normalmente se usaba como biblioteca para los sistemas operativos. Casi siempre estaban habilitadas localmente en los sistemas en los que operaban, aunque a veces pasaban mensajes entre las computadoras centrales.

Después de casi 30 años, las API se expandieron más allá de los entornos locales. A principios del año 2000, ya eran una tecnología importante para la integración remota de datos.



# API

Significa interfaz de programación de aplicaciones y es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.

Estas permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Permiten:

- Flexibilidad
- Simplificar el diseño
- Administrar y usar de mejor manera de las aplicaciones
- Proporcionar oportunidades de innovación, lo cual es ideal al momento de diseñar herramientas y productos nuevos.



# API

En las empresas las API se consideran como contratos, con documentación que representa un acuerdo entre las partes: si una de las partes envía una solicitud remota con cierta estructura en particular, esa misma estructura determinará cómo responderá el software de la otra parte.

Son un medio simplificado para conectar su infraestructura a través del desarrollo de aplicaciones nativas de la nube, pero también le permiten compartir sus datos con clientes y otros usuarios externos. Las API públicas representan un valor comercial único porque simplifican y amplían la forma en que se conecta con sus partners y, además, pueden rentabilizar sus datos (API's de Google).





# API Remotas

Están diseñadas para interactuar en una red de comunicaciones. Remoto indica que los recursos que administran las API están, fuera de la computadora que solicita los recursos.

Debido a que la red de comunicaciones más usada es Internet, la mayoría de las API están diseñadas de acuerdo con los estándares web. No todas las API remotas son API web, pero se puede suponer que las API web son remotas.

Las API web normalmente usan HTTP para solicitar mensajes y proporcionar una definición de la estructura de los mensajes de respuesta.

Por lo general, estos mensajes de respuesta toman la forma de un archivo XML o JSON, que son los formatos preferidos porque presentan los datos en una manera fácil de manejar para otras aplicaciones.



# API - Seguridad

Estas permiten habilitar el acceso a sus recursos y, al mismo tiempo, mantener la seguridad y el control.

La seguridad de las API tiene que ver con que se gestionen bien. Para conectarse a las API y crear aplicaciones que utilicen los datos o las funciones que estas ofrecen, se puede utilizar una plataforma de integración distribuida que conecte todos los elementos, incluidos los sistemas heredados y el Internet de las cosas (IoT).

# API - Seguridad

Respecto a las políticas de privacidad existen tres enfoques:

Privada: Las API solo se pueden usar internamente, así que las empresas tienen un mayor control sobre ellas. Esto le da a las empresas un mayor control sobre sus API.



Pública: Todos tienen acceso a las API, así que otras empresas pueden desarrollar API que interactúen con las de usted y así convertirse en una fuente de innovaciones. Esto permite que terceros desarrollen aplicaciones que interactúen con su API, y puede ser un recurso para innovar.



De partners: Las API se comparten con partners empresariales específicos, lo cual puede ofrecer flujos de ingresos adicionales, sin comprometer la calidad. Esto puede proporcionar flujos de ingreso adicionales, sin comprometer la calidad.

# SOAP Versus REST

## Protocolo de Acceso a Objetos Simples

- Es un protocolo
- Las API diseñadas con SOAP usan XML para el formato de sus mensajes y reciben solicitudes a través de HTTP o SMTP.
- Permite que las aplicaciones que funcionan en entornos distintos o están escritas en diferentes lenguajes compartan información.
- Cuenta con un estándar Oficial.

## RESTful

- Es un estilo de arquitectura.
- Normalmente usan JSON o XML para sus mensajes.
- Permite compartir información entre sistemas sin importar como esta implementado cada sistema independiente.
- Es más sencillo que un protocolo definido, no cuenta con un estándar oficial.
- Actualmente es el más utilizado por su facilidad.

# API Remotas - Arquitectura

Los enfoques más utilizados son SOA y microservicios.

Como vimos en el curso SOA es una mejora a las aplicaciones monolíticas y permite usar varias aplicaciones para que realicen diferentes funciones todo gracias a la orquestación y coreografía. Puede ser un riesgo si las aplicaciones no se comprenden claramente.

Las arquitecturas de microservicios descomponen las arquitecturas en partes más pequeñas. Estas pueden utilizar un marco de mensajes comunes como API RESTful que no necesita integración de capas adicionales. Cada servicio es independiente, puede ser reemplazado, mejorado o abandonado. Esta arquitectura optimiza los recursos y admite la escalabilidad dinámica de los servicios individuales.

# Application Programming Interface

Es una interfaz que permite compartir datos entre aplicaciones, podemos consumir los datos de otras Apps para nuestra App, por ej.





# Laboratorio

Sling

Recomendación usar  
<https://www.postman.com/>



# Laboratorio

Crear un catálogo de libros, debajo de cada libro crear editorial y un listado de autores. Se debe guardar los siguientes datos por tipo de evento:

- Autor (nombre, edad)
- Libro (título, autor, género, año)
- Editorial (nombre y país)

Crear 5 items de cada uno

Editar los recursos:

- Autor (actualizar nombre)
- Libro (actualizar campo año).

Eliminar

- Un autor

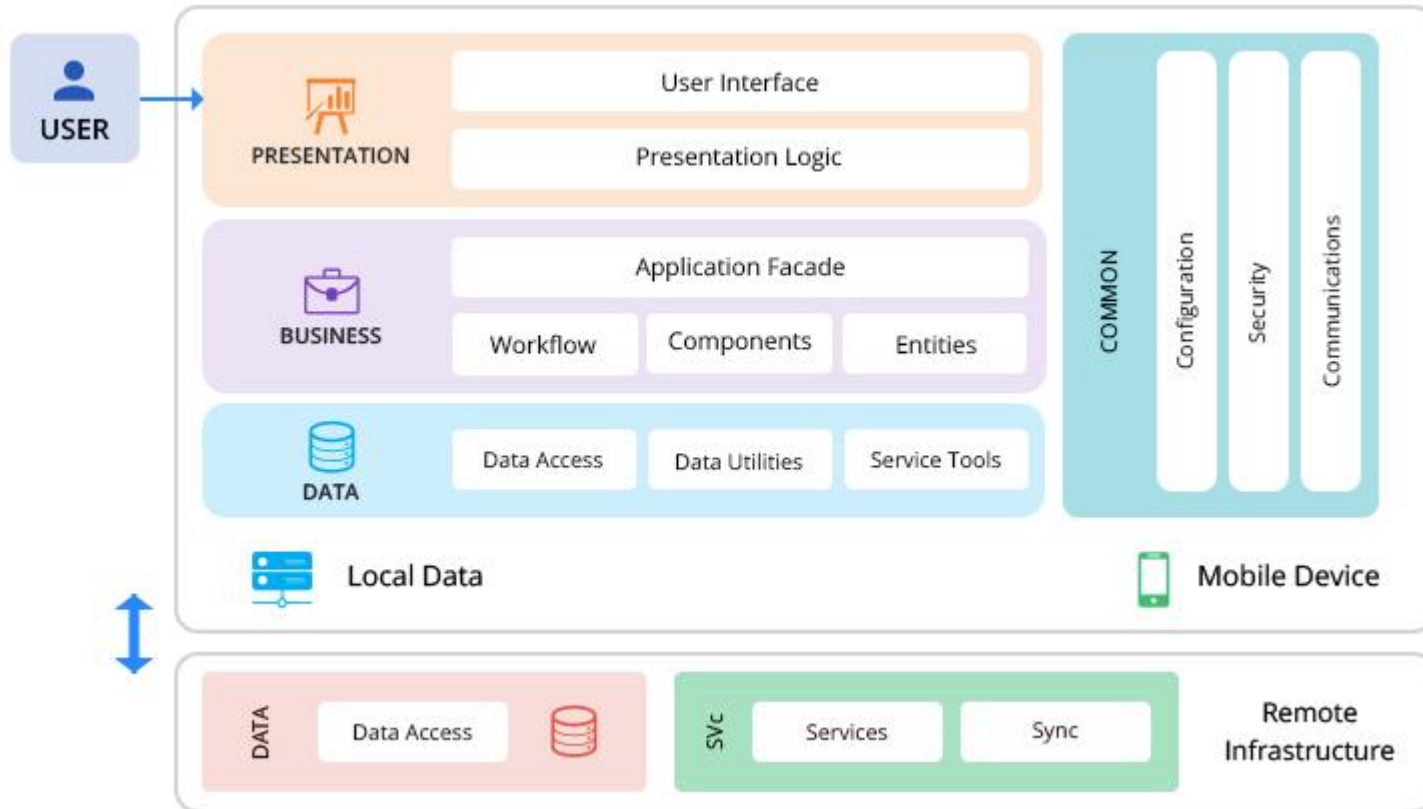




# **Arquitectura - Móviles**

## **Aplicaciones Híbridas**

# Analicemos la arquitectura...



# Aplicaciones Híbridas

Es una aplicación de software que combina elementos de aplicaciones nativas y aplicaciones web .

Las aplicaciones híbridas son esencialmente aplicaciones web que se han colocado en un shell de aplicaciones nativas. Una vez que se descargan de una tienda de aplicaciones y se instalan localmente, el shell puede conectarse a cualquier capacidad que la plataforma móvil brinde a través de un navegador que está integrado en la aplicación. El navegador y sus complementos se ejecutan en el back-end y son invisibles para el usuario final.

# Aplicaciones Híbridas

Las aplicaciones híbridas son populares porque permiten a los desarrolladores escribir código para una aplicación móvil una vez y aún acomodar múltiples plataformas (Android, iOS).

Debido a que las aplicaciones híbridas agregan una capa adicional entre el código fuente y la plataforma de destino, pueden funcionar un poco más lento que las versiones nativas o web de la misma aplicación.

# Aplicaciones Híbridas

Entre sus características principales están:

- La capacidad de funcionar independientemente de si el dispositivo está conectado o no.
- Integración con el sistema de archivos del dispositivo móvil.
- Integración con servicios web.
- Un navegador integrado para mejorar el acceso al contenido dinámico en línea.
- Opera en diferentes plataformas.

# ¿Cómo Funcionan?

Estas funcionan de manera similar a las aplicaciones web, pero al igual que las aplicaciones nativas, se descargan en el dispositivo. Al igual que las aplicaciones web, las aplicaciones híbridas generalmente se escriben en HTML5 , CSS y JavaScript .

Las aplicaciones híbridas ejecutan código dentro de un contenedor. El motor del navegador del dispositivo se utiliza para representar HTML y JavaScript y API nativas para acceder al hardware específico del dispositivo.

# ¿Cómo Funcionan?

Aunque una aplicación híbrida generalmente compartirá elementos de navegación similares a los de una aplicación web, si la aplicación puede funcionar sin conexión o no depende de sus funcionalidades.

Si una aplicación no necesita soporte de una base de datos, se puede hacer que funcione sin conexión.

# Aplicaciones Híbridas - Ventajas

Las ventajas están:

- Operará en diferentes plataformas.
- Tiempos de compilación más rápidos en comparación con las aplicaciones nativas.
- Más barato de desarrollar en comparación con la construcción de dos versiones de una aplicación nativa para dos plataformas diferentes.
- Más fácil de iniciar parches y actualizaciones.
- Puede trabajar en línea y sin conexión
- Usan web views.

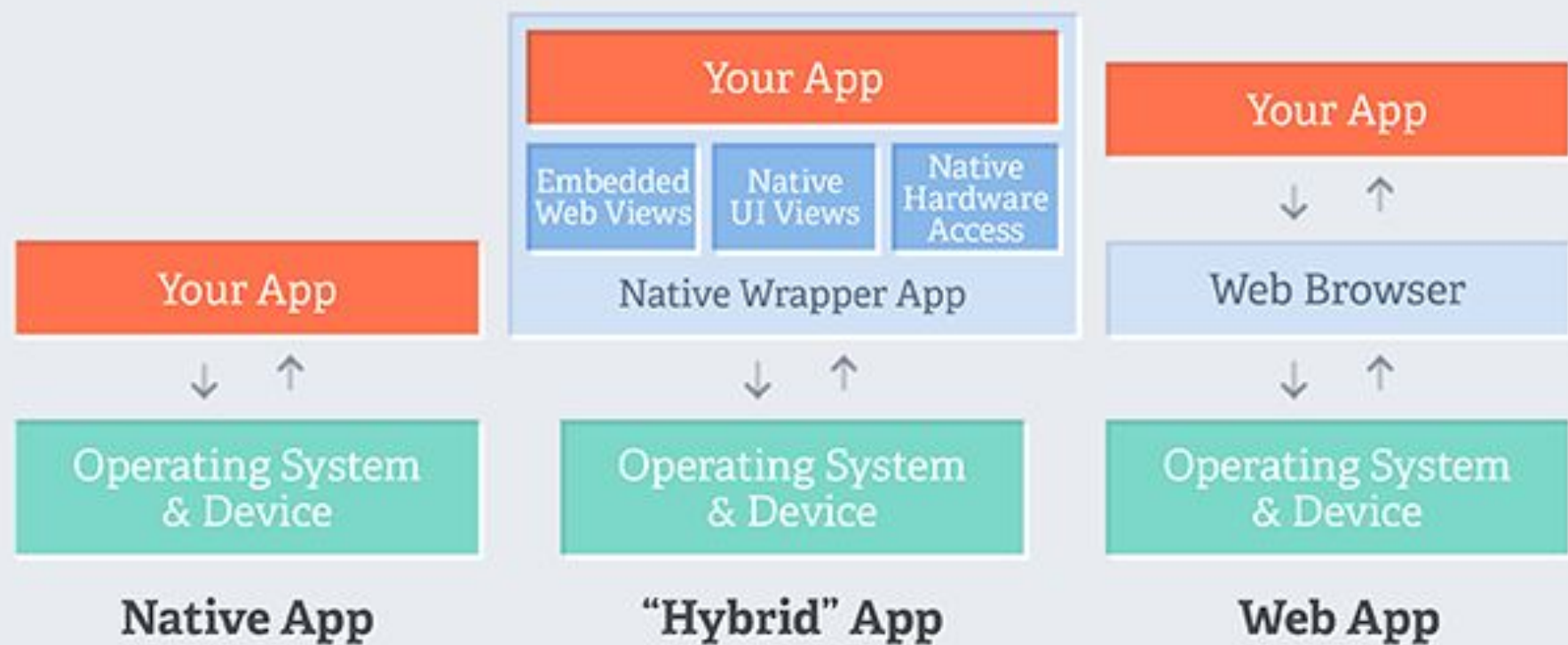


# Aplicaciones Híbridas - Desventajas

Las desventajas están:

- Variaciones debidas al desarrollo inclinado en una plataforma.
- La apariencia de una aplicación puede variar de una plataforma a otra.
- La experiencia del usuario ( UX ) puede disminuir si la interfaz de usuario ( UI ) no es similar y está suficientemente diseñada para los navegadores a los que está acostumbrado el usuario.
- Con respecto al control de calidad se debe probar la aplicación en una variedad de dispositivos para garantizar un funcionamiento correcto.
- Más difícil de ser aceptado y publicado en App Store

# Mobile App Technology Stacks

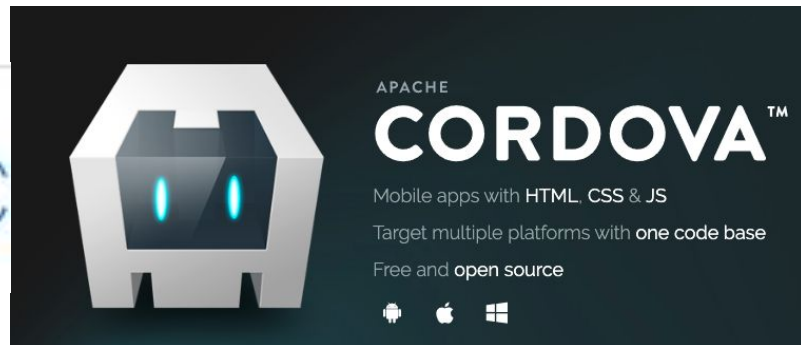


# Híbrido Vs Nativo Vs Web

Híbrido	Nativo	Web
<ul style="list-style-type: none"><li>• Se crea para ser utilizado en múltiples plataformas.</li><li>• Solo pueden aprovechar parte del hardware de un dispositivo.</li><li>• Escrito bajo utilizando lenguajes utilizados para crear aplicaciones web.</li><li>• Se instala en el dispositivo.</li><li>• Es más lenta y la experiencia de usuario es estándar.</li><li>• Aparecen en el store/market.</li></ul>	<ul style="list-style-type: none"><li>• Se crean específicas para una plataforma.</li><li>• Pueden aprovechar el hardware de un dispositivo.</li><li>• Escritas bajo el lenguaje de programación del sistema operativo.</li><li>• Se instala en el dispositivo</li><li>• Es más rápida y tiene una mejor experiencia de usuario.</li><li>• Aparecen en el store/market.</li></ul>	<ul style="list-style-type: none"><li>• Se crean para funcionar dentro de un browser.</li><li>• Puede aprovechar sólo los recursos proporcionados al browser.</li><li>• Escritas utilizando lenguajes como HTML, librerías de javascript, css, etc.</li><li>• No se instala en el dispositivo.</li><li>• Pueden ser lentas, menos intuitivas.</li><li>• No aparecen en el store/market.</li></ul>

**En Resumen:** Las aplicaciones híbridas combinan aplicaciones nativas y web. Esta se instala y funciona de manera similar a una aplicación nativa, pero tiene el funcionamiento interno de una aplicación web.

# Herramientas





# **Arquitectura - Móviles**

**Cross Compiled / Cross Platform**



# Cross Compiled / Cross Platform

Este es el nivel más cercano al desarrollo nativo, pero tiene la ventaja de usar una única base de código para apuntar a Android e iOS.

Con este tipo de herramientas se crea el código que luego es compilado hacia la plataforma nativa.

Generando una aplicación Nativa con algunos pequeños cambios.



# Cross Compiled - Ventajas

Entre las ventajas están:

- Se obtiene una app para ambas plataformas con un solo lenguaje.
- Aproximadamente el mismo rendimiento que las aplicaciones nativas, ya que también tratan con componentes nativos de la interfaz de usuario

Nota: Flutter, proporcionará su propio conjunto de widgets junto con el código de su aplicación



# Cross Compiled - Desventajas

Entre las desventajas están:

- Soporte ligeramente retrasado para las últimas actualizaciones de la plataforma.
- Código no compatible con desarrollo web.
- Incluso con una única base de código, el desarrollador debe conocer los componentes nativos.



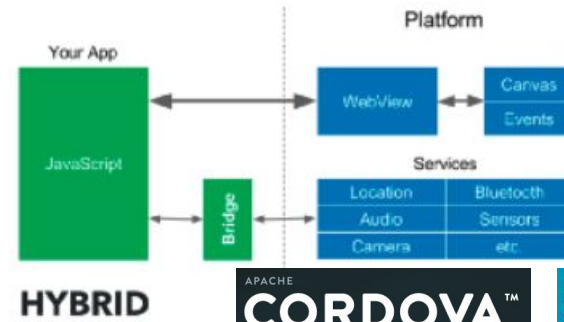
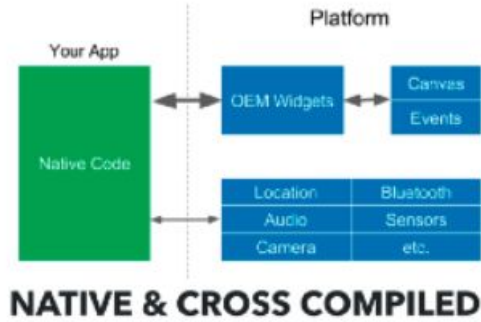
# Herramientas



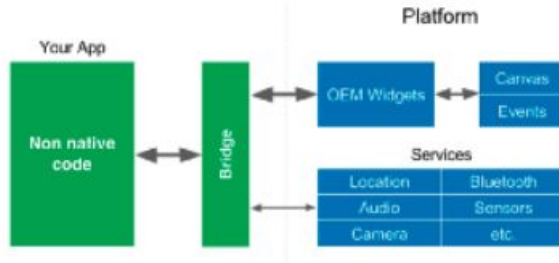
## Xamarin

Free. Cross-platform. Open source.

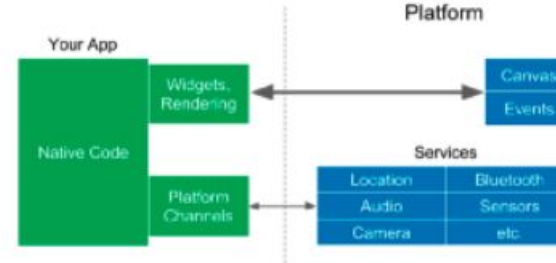
An app platform for building Android and iOS apps with .NET and C#.



## WEB NATIVE & CROSS COMPILED



## FLUTTER



Cuatro arquitecturas de tiempo de ejecución de aplicaciones móviles diferentes.

Imagen tomada del siguiente [link](#)

# Laboratorio



Flutter

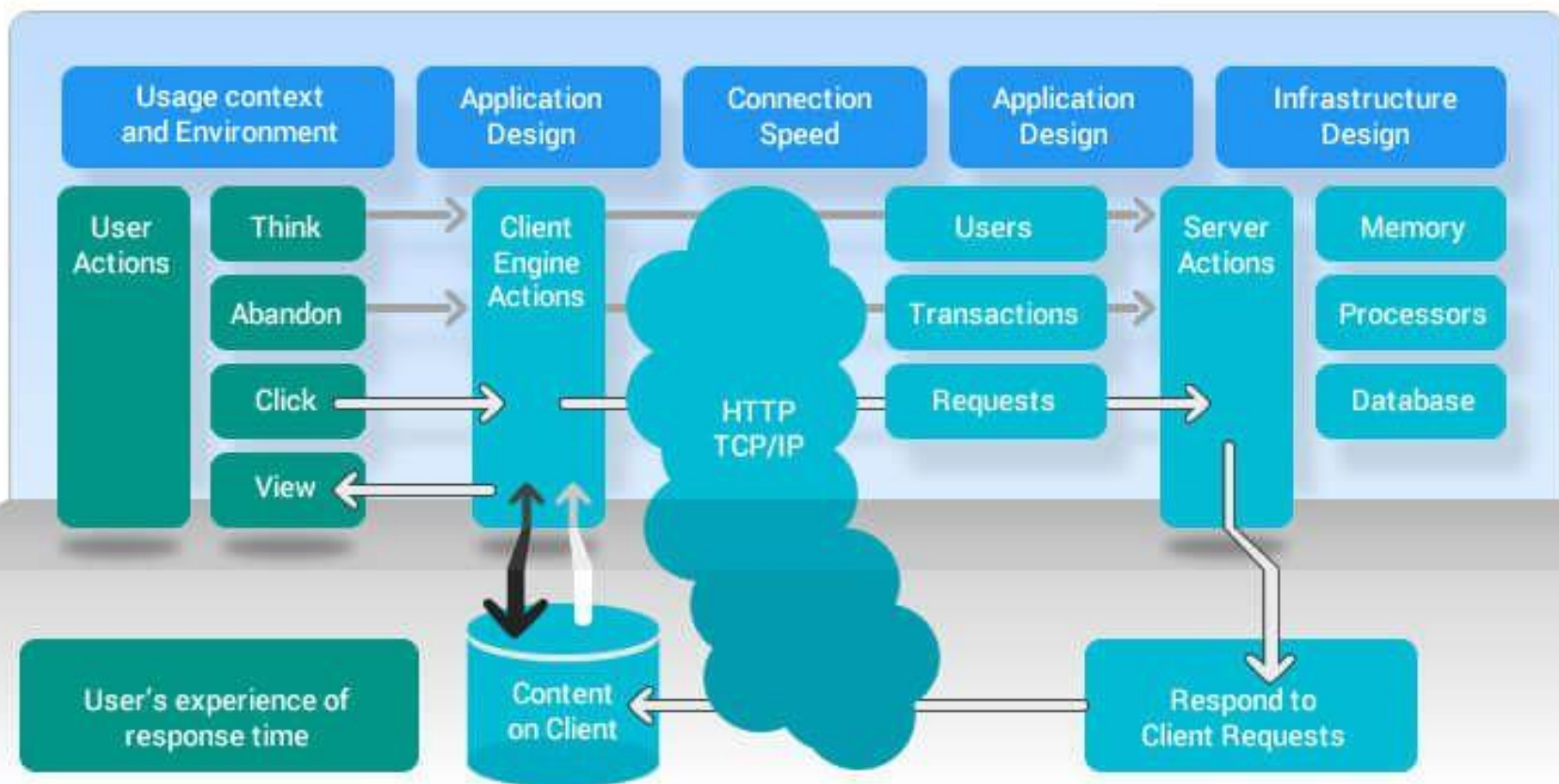
First Flutter App: Building  
your first Flutter app |  
Workshop



# **Modelo y Diseño de Arquitectura**

## **- Web**

### **Rich Internet Application (RIA)**



**The Rich Internet Application Behaviour Model**

# RIA (Rich Internet Application)

Las aplicaciones de Internet enriquecidas son aplicaciones web que incorporan la mayoría de las características de una aplicación de escritorio al tiempo que proporcionan una interfaz de usuario que responde mejor que una aplicación tradicional.

A Diferencia de las aplicaciones tradicionales, estas permiten tener actualizada la aplicación, permitiendo al usuario tener los features.

Las aplicaciones de Internet enriquecidas se usan comúnmente para ofrecer funciones como videos, procesadores de texto y juegos en línea a los usuarios.

# RIA (Rich Internet Application)

Las aplicaciones RIA permiten trasladar parte del cálculo al cliente. Los elementos del programa en el cliente están listos para recibir y ejecutar comandos asíncronos del servidor. Eliminando request adicionales innecesarios.

Normalmente en RIA la interfaz comprende una sola pagina que comprende subpáginas, administra las interacciones del usuario, como en una aplicación de escritorio. Este paradigma evita actualizaciones de página completa en cada interacción y permite que las aplicaciones carguen, muestran y actualicen elementos de página individuales de forma independiente.

# RIA - Beneficios

Los RIA ofrecen a las organizaciones una forma comprobada y rentable de ofrecer altos niveles de funcionalidad y brindar beneficios comerciales reales:

- Ofrecen a los usuarios una experiencia más rica y atractiva y aumenta la productividad del usuario.
- Admite la visualización avanzada de datos, incluidos cuadros y presentaciones gráficas de datos.
- Permite tener la información en tiempo real.
- Heredan los beneficios de las aplicaciones web tales como accesibilidad, escalabilidad y portabilidad.



# RIA - Ventajas

En comparación con las aplicaciones estándar basadas en HTML, los RIA potencian su negocio con soluciones que son más:

- **Consistente:** a diferencia de las aplicaciones HTML que a menudo necesitan recargar varias páginas para completar una acción, los RIA permiten transiciones de etapas sin interrupciones y controladas interactivamente que no distraen a los usuarios del objetivo final de una acción.
- **Orientado:** las tecnologías de RIA brindan una excelente experiencia para el desarrollo de interfaces de usuario atractivas que se centran por completo en sus necesidades y las de sus clientes, y tienen el diseño de la aplicación optimizado para el propósito que sirve.

# RIA - Ventajas

En comparación con las aplicaciones estándar basadas en HTML, los RIA potencian su negocio con soluciones que son más:

- **Receptivo:** debido a las tecnologías específicas utilizadas, los RIA simplifica procesos complejos (como el registro o la compra), ahorran ancho de banda y funcionan considerablemente más rápido que las aplicaciones tradicionales.
- **Inteligente:** ajustando dinámicamente su comportamiento mediante la captura, el mantenimiento y el uso de información contextual, los RIA entregan de forma interactiva a los usuarios exactamente lo que necesitan y reducen su carga de trabajo, tiempo y esfuerzo dedicados al desempeño de acciones específicas.

# Aplicaciones Web Tradicional Versus RIA

## Aplicaciones Web Tradicionales

- Basada en Texto
- Comunicación Sincrona: cada vez que se solicita algo se espera la respuesta del servidor.
- Necesita recargar la página o cambiar de página.
- La carga de trabajo la tiene el servidor (sesión de usuario, procesar los datos).
- Cliente liviano regularmente.
- La lógica de presentación y de negocio suelen estar dentro de la misma capa física, por lo que la comunicación entre ambas no supone ninguna complejidad.

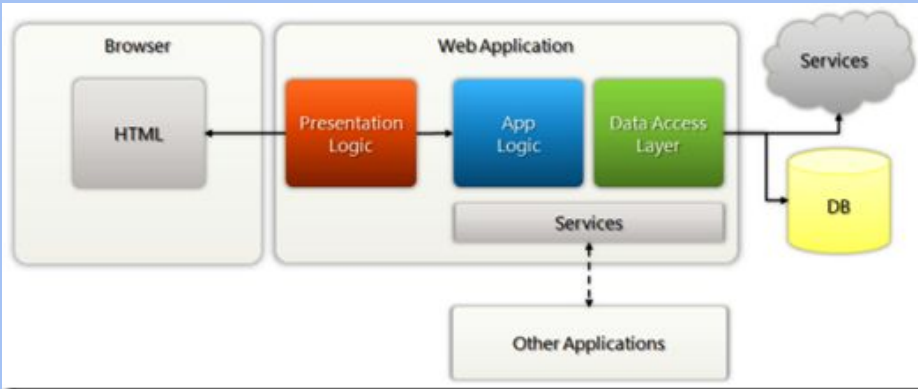
## RIA

- Basados en la interacción del cliente.
- Comunicación asíncrona: normalmente el cliente puede realizar varias peticiones sin esperar la respuesta.
- Normalmente se actualiza solo la porción de información que se necesita.
- Aprovecha también los recursos del cliente, ya que ejecuta la mayoría de funciones localmente.
- El cliente es pesado, debido a que tenemos parte de la aplicación en el.
- La lógica de presentación y negocio no están en la misma capa física, hay que considerar proxys, validación y autenticación.

# Aplicaciones Web Tradicional Versus RIA

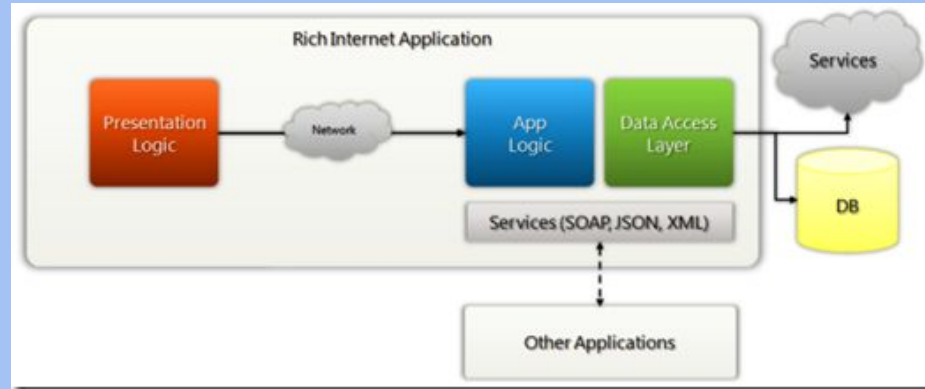
## Aplicaciones Web Tradicionales

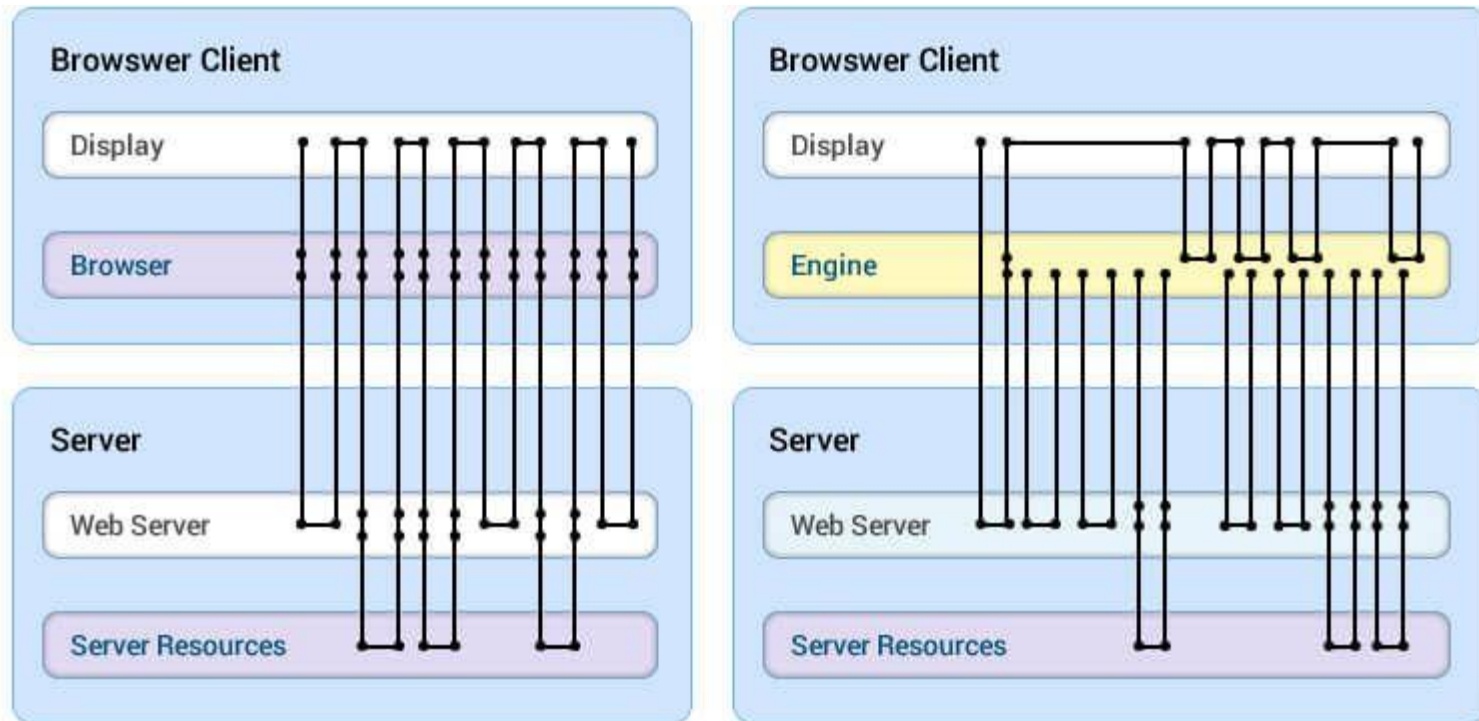
- Arquitectura base



## RIA

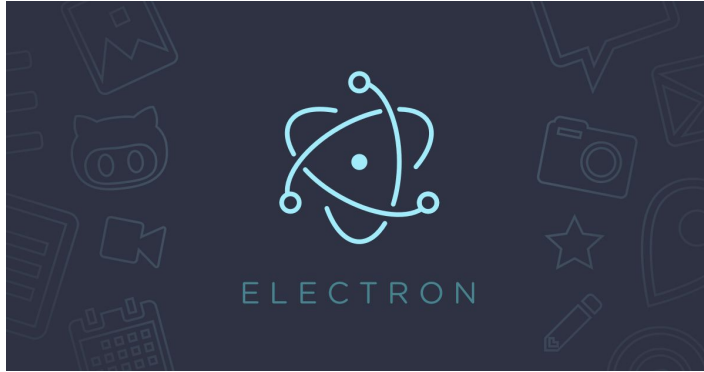
- Arquitectura base





The Communication Model: Traditional Web Application vs. Rich Internet Application

# RIA Herramientas

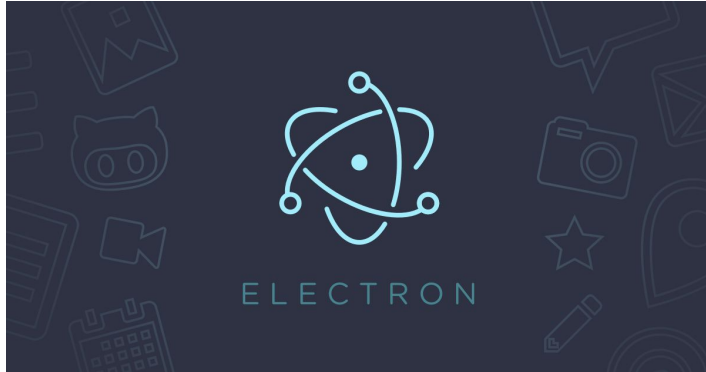


# Algunas RIA



**Basecamp 3**

# Laboratorio



Electron - Hello World :  
<https://www.electronjs.org/docs/tutorial/quick-start>





# **Modelo y Diseño de Arquitectura - Web**

## **Single Page Application (SPA)**



# SPA (Single Page Application)

Las aplicaciones de una sola página (SPA) son aplicaciones web que cargan una sola página HTML y actualizan dinámicamente esa página a medida que el usuario interactúa con la aplicación.

Los SPA utilizan AJAX y HTML5 para crear aplicaciones web fluidas y receptivas, sin recargas constantes de página. Sin embargo, esto significa que gran parte del trabajo ocurre en el lado del cliente, en JavaScript. Afortunadamente, hay muchos frameworks de JavaScript de que facilitan la creación de SPA.

# Aplicaciones Web Tradicional Versus SPA

## Aplicaciones Web Tradicionales

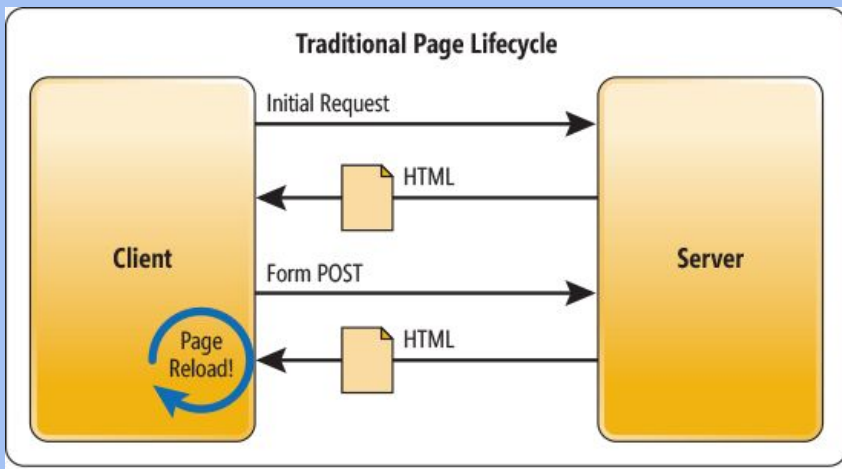
- cada vez que la aplicación llama al servidor, el servidor presenta una nueva página HTML. Esto desencadena una actualización de la página en el navegador.
- Actualización de la página se da por recarga de toda la página.
- Aplicaciones no son tan fluidas.
- Puede no hacerse la separación de presentación y lógica.
- Se realizan muchas llamadas al servidor.
- Cliente y servidor dependientes.

## SPA

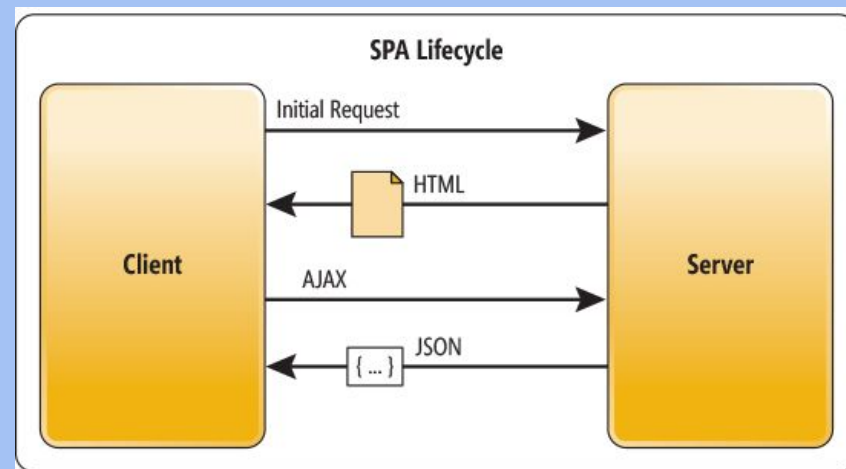
- después de cargar la primera página, toda interacción con el servidor ocurre a través de llamadas AJAX. Estas llamadas AJAX devuelven datos, no marcas, generalmente en formato JSON.
- Se actualiza solo un un segmento de la página de forma dinámica.
- Aplicaciones más fluidas y receptivas.
- Separación entre presentación (HTML) y lógica (respuestas json).
- Se optimiza el número de llamadas al servidor y los recursos.
- Cliente y servidor independientes, el servidor actúa como capa de servicio tras la carga inicial de la página.

# Aplicaciones Web Tradicional Versus SPA

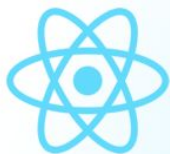
## Aplicaciones Web Tradicionales



## SPA



# SPA Herramientas



ReactJS



Vue.js

# Laboratorio



ReactJS

React - Tutorial :

<https://reactjs.org/tutorial/tutorial.html>



# Arquitectura de sistemas, comportamiento y optimización

Week 10

M.Sc. Juan Luis Flores Pineda

# Agenda

## → Arquitectura Web

- ◆ Incidencias comunes.
- ◆ Incidencias de aplicación
- ◆ Incidencias de seguridad
- ◆ Herramientas
- ◆ Análisis de casos de arquitectura.

## → Introducción a la Integración de software

- ◆ Introducción.
- ◆ Entrega y Implementación continua
- ◆ Diferencias entre aplicaciones distribuidas y aplicaciones integradas.

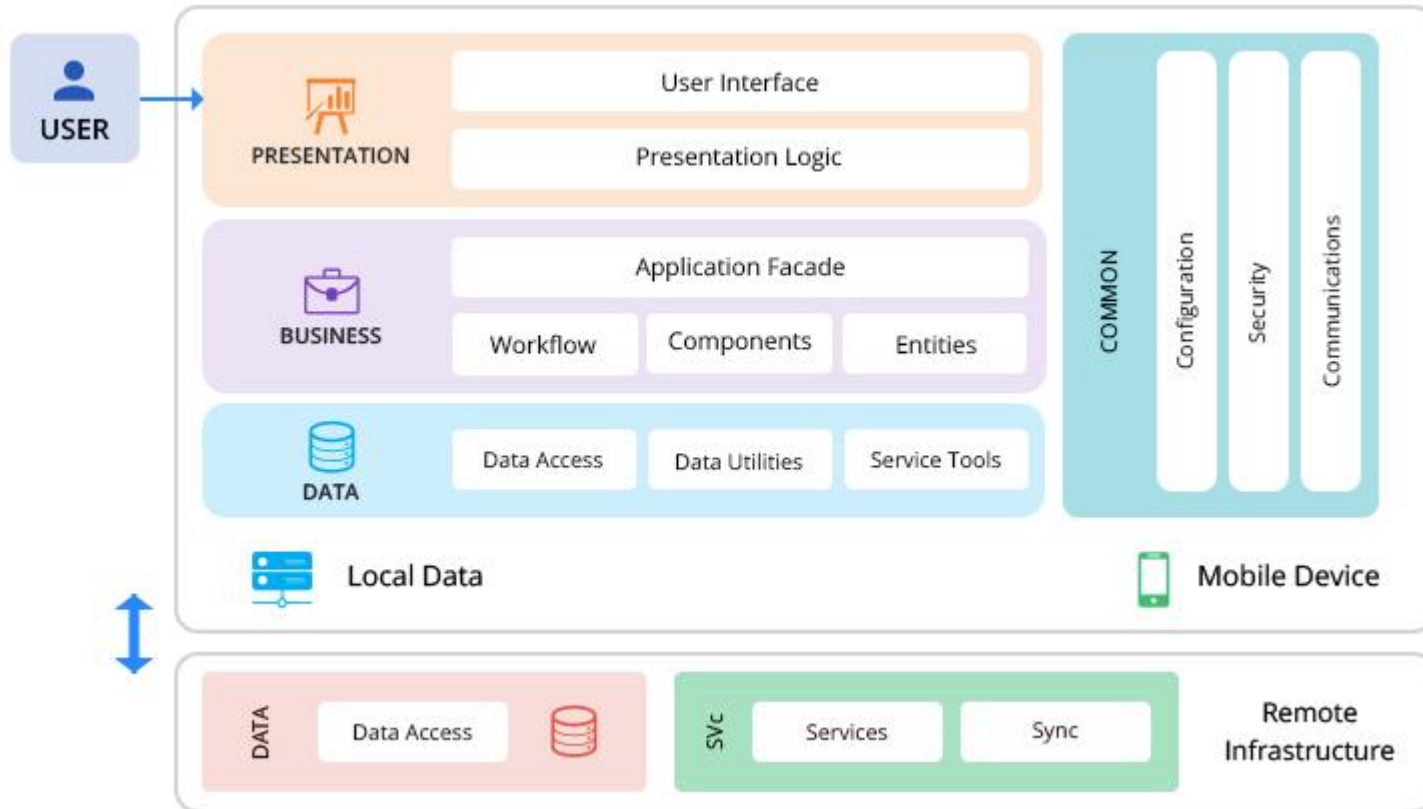




# Arquitectura Web

## Incidencias Comunes

# Analicemos la arquitectura...




# Introducción

Una de las tareas principales para el arquitecto de soluciones es hacer que dicho sistema sea altamente independiente de: proveedor, base de datos, lenguaje de programación, marco, etc. Para lograrlo, necesita un gran esfuerzo para invertir. Pregúntese como ingeniero, ¿cuántas veces en su carrera migró el motor de la base de datos de su solución a otro? ¿Alguna vez reemplazó su base de datos MySQL con Oracle, por ejemplo?

# Introducción

Para lograr la independencia, debe invertir mucho tiempo y dinero. Si es una startup que necesita probar su MVP en el mercado, nunca debe construirla teniendo en cuenta dicha arquitectura. Pero una vez que MVP se convierte en un producto que es utilizado por un número significativo de personas, la arquitectura de MVP debe ser reemplazada por una que sea escalable, flexible, independiente, etc. Esta trampa es muy común, por lo que puede diseñar demasiado su solución en el comienzo, lo que destruye su capacidad de ir al mercado rápidamente o nunca reemplaza su arquitectura MVP con la adecuada.



"Un buen arquitecto de soluciones tratará de comprender el futuro del sistema y, basándose en eso, elegirá el nivel correcto de arquitectura que debería estar en su lugar."



# Incidencias de los Fisicas

# Incidencias Fisicas

Son las incidencias que nacen debido a cuestiones físicas tales como:

- Capacidad del servidor
  - Disco
  - Memoria
  - Conexión
- Ubicación
  - Tiempo de respuesta
  - Conexión entre distintos servidores.
- Ancho de banda
  - Protocolos de Internet



# Incidencias de aplicación



# Escalabilidad

Debido al desarrollo de alta velocidad y a la idea de tener un MVP lo antes posible. Esto ha causado que la mayoría de de equipos den por sentado la escalabilidad.

Muchas veces seleccionar una base de datos escalable y un servidor web e intentar separar todas las capas de aplicaciones.

Debemos pensar que sucede al querer escalar la aplicación que problemas podemos encontrarnos, que sucede al poner un load balancer, como sincronizamos los distintos servidores.

# Escalabilidad

Entre los problemas que podemos tener en escalabilidad están:

1. Sincronización de los distintos servidores.
2. Tiempos de respuesta entre servidores.
3. Diferentes Ubicaciones.
4. Módulos difíciles de escalar o adaptar.
5. Desarrollo no pensando en ser escalable.
6. Otros.

# SEO incorrecto o faltante

Se debe pensar en SEO desde el inicio de la aplicación, en muchas ocasiones se piensa en esta actividad al final.

Ya que está relacionado con la buena configuración del contenido, palabras clave (keywords), etiquetas alternativas para las imágenes, mapa del sitio, estructura del contenido, tiempos de carga eficientes, enlaces inteligentes.

Solución: muchas veces reorganizar el contenido completo, agregar implementación.



# Incidencias de Seguridad



# Seguridad - Introducción

La seguridad es un problema que vemos cada vez más frecuente en todo tipo de arquitectura, se ha vuelto muy común que hackers roban datos de empresas importantes (Sony, etc).

Es Importante siempre practicar medidas de seguridad y siempre esté preparado para protegerse y proteger el futuro de su empresa de un ataque del que nunca se recuperará.

La mejor manera de saber si su sitio web o servidor es vulnerable es realizar auditorías de seguridad periódicas.

Entre los casos más comunes están:



# Seguridad - La inyección SQL

Es un tipo de vulnerabilidad de seguridad de la aplicación web en la que un atacante intenta usar el código de la aplicación para acceder o dañar el contenido de la base de datos. Si tiene éxito, esto permite al atacante crear, leer, actualizar, alterar o eliminar datos almacenados en la base de datos de fondo.

La inyección SQL es uno de los tipos más frecuentes de vulnerabilidades de seguridad de aplicaciones web .



# Seguridad - Cross Site Scripting (XSS)

Las secuencias de comandos entre sitios (XSS) se dirigen a los usuarios de una aplicación inyectando código, generalmente una secuencia de comandos del lado del cliente como JavaScript, en la salida de una aplicación web.

El concepto de XSS es manipular los scripts del lado del cliente de una aplicación web para ejecutarlos de la manera deseada por el atacante. XSS permite a los atacantes ejecutar scripts en el navegador de la víctima que pueden secuestrar sesiones de usuario, desfigurar sitios web o redirigir al usuario a sitios maliciosos.



# Seguridad - Mal Manejo de Autenticación

La autenticación y la administración de sesión interrumpidas abarcan varios problemas de seguridad, todos ellos relacionados con el mantenimiento de la identidad de un usuario.

Si las credenciales de autenticación y los identificadores de sesión no están protegidos en todo momento, un atacante puede secuestrar una sesión activa y asumir la identidad de un usuario.





# Seguridad - Exposición de referencias

La referencia directa a objetos inseguros es cuando una aplicación web expone una referencia a un objeto de implementación interno.

Los objetos de implementación interna incluyen archivos, registros de bases de datos, directorios y claves de bases de datos.

Cuando una aplicación expone una referencia a uno de estos objetos en una URL, los hackers pueden manipularla para obtener acceso a los datos personales de un usuario.



# Seguridad - Configuración incorrecta de Seguridad

La configuración incorrecta de seguridad abarca varios tipos de vulnerabilidades, todas centradas en la falta de mantenimiento o la falta de atención a la configuración de la aplicación web.

Se debe definir e implementar una configuración segura para la aplicación, los marcos, el servidor de aplicaciones, el servidor web, el servidor de bases de datos y la plataforma.

La configuración incorrecta de seguridad brinda a los hackers acceso a datos o funciones privadas y puede dar lugar a un compromiso completo del sistema.



# Seguridad - Cross-Site Request Forgery (CSRF)

Cross-Site Request Forgery (CSRF) es un ataque malicioso en el que se engaña a un usuario para que realice una acción que no tenía la intención de hacer.

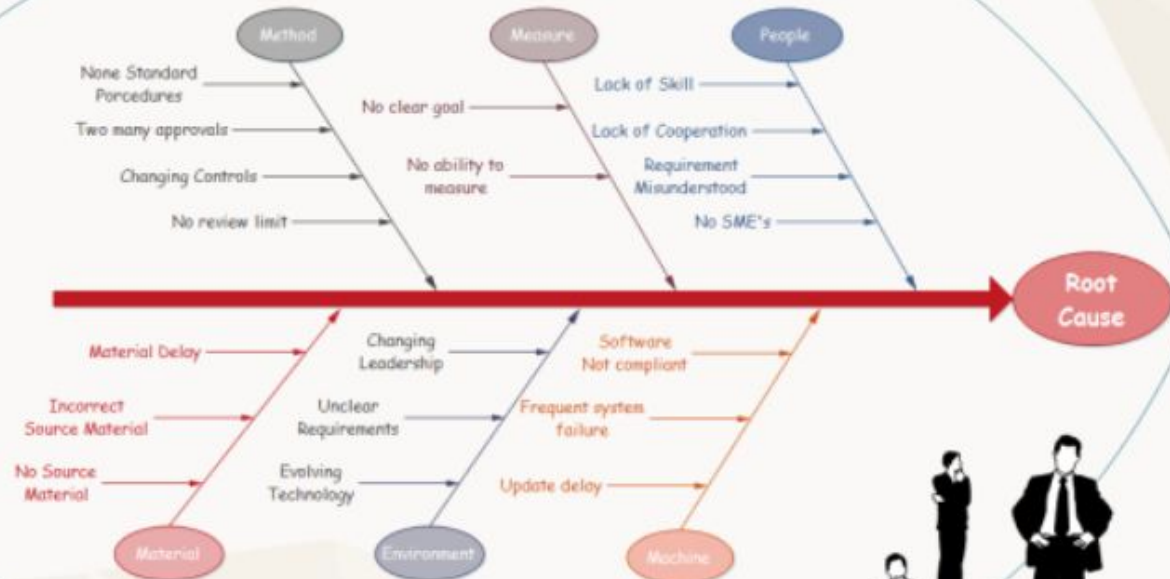
Un sitio web de terceros enviará una solicitud a una aplicación web con la que el usuario ya está autenticado (por ejemplo, su banco). El atacante puede acceder a la funcionalidad a través del navegador ya autenticado de la víctima. Los objetivos incluyen aplicaciones web como redes sociales, en clientes de correo electrónico del navegador, banca en línea e interfaces web para dispositivos de red.



# Arquitectura Web

## Técnica de Análisis

## Root Cause Analysis



# Análisis de Causa Raíz

El análisis de causa raíz (RCA) es una técnica popular y de uso frecuente que ayuda a las personas a responder la pregunta de por qué ocurrió el problema en primer lugar. Busca identificar el origen de un problema utilizando un conjunto específico de pasos, con herramientas asociadas, para encontrar la causa principal del problema, de modo que pueda:

1. Determina lo que pasó.
2. Determina por qué sucedió.
3. Averigüe qué hacer para reducir la probabilidad de que vuelva a suceder.

# Análisis de Causa Raíz

RCA supone que los sistemas y eventos están interrelacionados. Una acción en un área desencadena una acción en otra, y en otra, y así sucesivamente.

Al rastrear estas acciones, puede descubrir dónde comenzó el problema y cómo se convirtió en el síntoma que enfrenta ahora.

RCA analiza los tres tipos de causas. Implica investigar los patrones de efectos negativos, encontrar fallas ocultas en el sistema y descubrir acciones específicas que contribuyeron al problema. Esto a menudo significa que RCA revela más de una causa raíz.

# Análisis de Causa Raíz - Tipos de Causas

Por lo general, encontrará tres tipos básicos de causas:

1. Causas físicas : los elementos materiales tangibles fallaron de alguna manera.
2. Causas humanas : las personas hicieron algo mal o no hicieron algo que era necesario. Las causas humanas generalmente conducen a causas físicas.
3. Causas organizativas : un sistema, proceso o política que las personas usan para tomar decisiones o hacer su trabajo es defectuoso.



# Análisis de Causa Raíz

**Dificultad:** Determinar qué tan lejos llegar en su investigación requiere buen juicio y sentido común. Teóricamente, podría seguir rastreando las causas fundamentales, pero el esfuerzo no serviría para nada.

# Análisis de Causa Raíz - Paso 1

RCA tiene cinco pasos identificables:

1. Definir el problema:
  - Responde a las preguntas
    - i. ¿Qué ves que pasa?
    - ii. ¿Cuáles son los síntomas específicos?

# Análisis de Causa Raíz - Paso 2

2. Recopilar los datos:
  - a. Responde a las preguntas:
    - i. ¿Qué prueba tienes de que el problema existe?
    - ii. ¿Cuánto tiempo ha existido el problema?
    - iii. ¿Cuál es el impacto del problema?

Debe analizar completamente una situación antes de poder seguir para ver los factores que contribuyeron al problema. Para maximizar la efectividad de su RCA, reúna a todos, expertos y los que están más familiarizadas con el problema.

# Análisis de Causa Raíz - Paso 3

3. Identifique los posibles factores causales:
  - a. Responde a las preguntas
    - i. ¿Qué secuencia de eventos conduce al problema?
    - ii. ¿Qué condiciones permiten que ocurra el problema?
    - iii. ¿Qué otros problemas rodean la ocurrencia del problema central?

Durante esta etapa, identifique tantos factores causales como sea posible. Con demasiada frecuencia, las personas identifican uno o dos factores y luego se detienen, pero eso no es suficiente. Con RCA, no desea tratar simplemente las causas más obvias: desea profundizar.

# Análisis de Causa Raíz - Paso 3

Use estas herramientas para ayudar a identificar factores causales:

- 5 porqués - ¿Pregunta porque?" hasta llegar a la raíz del problema.
- Profundizar - Divida un problema en partes pequeñas y detalladas para comprender mejor el panorama general.
- Apreciación - Usa los hechos y pregunta "¿Y qué?" para determinar todas las posibles consecuencias de un hecho.
- Diagramas de causa y efecto - Cree una tabla de todos los posibles factores causales, para ver dónde puede haber comenzado el problema.

# Análisis de Causa Raíz - Paso 4

4. Identifique la (s) causa (s) raíz (s)
  - a. Responde a las preguntas:
    - i. ¿Por qué existe el factor causal?
    - ii. ¿Cuál es la verdadera razón por la que ocurrió el problema?

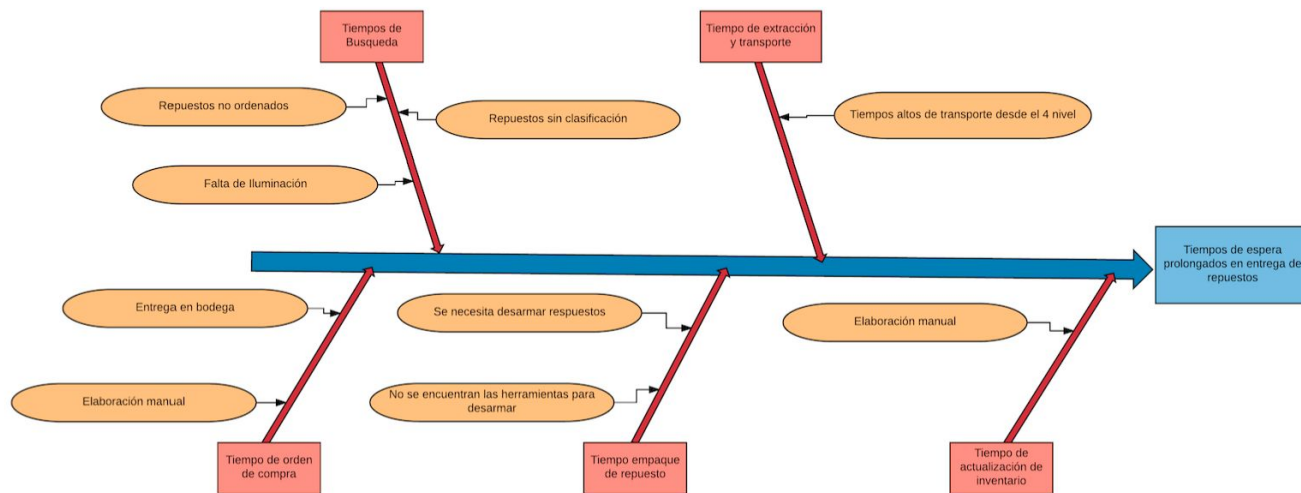
Use las mismas herramientas que usó para identificar los factores causales (*paso tres*) para observar las raíces de cada factor. Estas herramientas están diseñadas para alentarle a profundizar en cada nivel de causa y efecto.

# Análisis de Causa Raíz - Paso 5

5. Recomendación e implementar soluciones
  - a. Responde a las preguntas:
    - i. ¿Qué puede hacer para evitar que el problema vuelva a ocurrir?
    - ii. ¿Cómo se implementará la solución?
    - iii. ¿Quién será responsable de ello?
    - iv. ¿Cuáles son los riesgos de implementar la solución?

Analice su proceso de causa y efecto e identifique los cambios necesarios para varios sistemas. También es importante que planifique con anticipación para predecir los efectos de su solución. De esta manera, puede detectar posibles fallas antes de que sucedan.

# Ejemplo:





Ejemplo:

# Utilizando 5 Porqués



# Arquitectura Web

## Herramientas

# Herramientas Velocidad - Página Web



PageSpeed Insights

HOME

DOCS

Mejora la velocidad de tus páginas web en todos los dispositivos

Escribe una URL de página web

ANALIZAR

## Novedades

Read the latest [Google Webmaster posts](#) about performance & speed.

## Enviar comentarios

¿Tienes alguna pregunta concreta y rápida sobre PageSpeed Insights? Hazla en [Stack Overflow](#). Si lo que quieres es enviar comentarios más generales, crea una conversación en nuestra [lista de distribución](#).

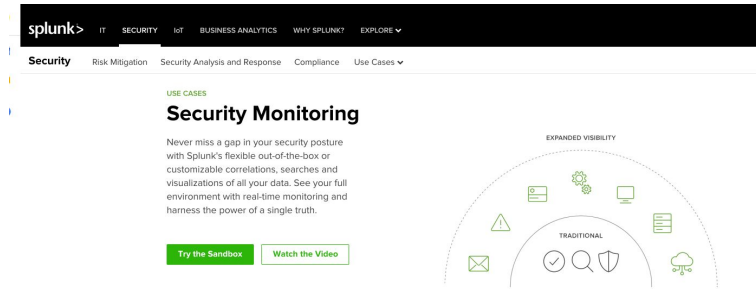
## Rendimiento web

Más información sobre [herramientas de rendimiento web de Google](#).

## Sobre PageSpeed Insights

PageSpeed Insights analiza el contenido de una página web y, a continuación, genera sugerencias para mejorar la velocidad de las páginas. [Más información](#).

# Herramientas Monitoreo



# Herramientas - Test de Stress





# Arquitectura Web

Ejercicio - Análisis de casos

## Pagina Web

Realice el RCA aplicado a la página de la Universidad utilizando el diagrama de causa y Efecto. Utilice page speed Insights.



# Ejercicio

Instale JMeter  
Pruebas de stress a google.com  
100 usuarios (threads)





# Introducción a la Integración de Software

## Introducción

# Integración Continua



Esta es una práctica a raíz de las metodologías ágiles, en donde el software se empezó a automatizar, debido a los tiempos de implementación de entrega continua. Esto dio nacimiento a la integración continua.

La integración continua es una práctica de desarrollo de software en la que los desarrolladores fusionan regularmente sus cambios de código en un repositorio central, después de lo cual se ejecutan compilaciones y pruebas automatizadas.

# Integración Continua

La integración continua se refiere con mayor frecuencia a la etapa de compilación o integración del proceso de lanzamiento de software e implica tanto un componente de automatización como un componente cultural del equipo de desarrollo.

Los objetivos clave de la integración continua son:

- encontrar y abordar los errores más rápido,
- mejorar la calidad del software y
- reducir el tiempo que lleva validar y
- lanzar nuevas actualizaciones de software.

# Integración Continua

En el pasado, los desarrolladores de un equipo podían trabajar de forma aislada durante un período prolongado de tiempo y solo fusionar sus cambios en la rama maestra una vez que se completaba su trabajo.

Esto hizo que los cambios de código de fusión fueran difíciles y llevaran mucho tiempo, y también resultó en la acumulación de errores durante mucho tiempo sin corrección.

Estos factores dificultaron la entrega rápida de actualizaciones a los clientes.

# Integración Continua - ¿Cómo Funciona?

Con una integración continua, los desarrolladores frecuentemente se comprometen a un repositorio compartido utilizando un sistema de control de versiones como Git.

Antes de cada confirmación, los desarrolladores pueden elegir ejecutar pruebas unitarias en su código como una capa de verificación adicional antes de la integración.

Un servicio de integración continua crea y ejecuta automáticamente pruebas unitarias en los nuevos cambios de código para detectar inmediatamente cualquier error.

# Integración Continua - ¿Cómo Funciona?

Con la entrega continua , los cambios de código se crean, prueban y preparan automáticamente para su lanzamiento a producción.

La entrega continua se expande tras la integración continua mediante la implementación de todos los cambios de código en un entorno de prueba y/o en un entorno de producción después de la etapa de desarrollo.

# Integración continua - Beneficios

## Buscar y solucionar errores más

**rápido:** Con pruebas más frecuentes, su equipo puede descubrir y abordar errores antes de que se conviertan en problemas más grandes más adelante.



**Entregar actualizaciones más rápido:** La integración continua ayuda a su equipo a entregar actualizaciones a sus clientes de manera más rápida y frecuente.



## Mejora la productividad del

**desarrollador:** debido a que ayuda a su equipo a ser más productivo al liberar a los desarrolladores de las tareas manuales y al fomentar comportamientos que ayudan a reducir la cantidad de errores y errores lanzados a los clientes.





# **Introducción a la Integración de Software**

**Aplicaciones Distribuidas**



# Aplicaciones Distribuidas

Son aplicaciones o software que se ejecutan en varias computadoras dentro de una red al mismo tiempo y se pueden almacenar en servidores o con computación en la nube.

A diferencia de las aplicaciones tradicionales que se ejecutan en un solo sistema, las aplicaciones distribuidas se ejecutan en múltiples sistemas simultáneamente para una sola tarea o trabajo.

# Aplicaciones Distribuidas

Estas pueden comunicarse con múltiples servidores o dispositivos en la misma red desde cualquier ubicación geográfica.

La naturaleza distribuida de las aplicaciones se refiere a la distribución de datos en más de una computadora en una red.

Las aplicaciones distribuidas se dividen en dos programas separados:

1. Software del cliente
2. Software del servidor.

# Aplicaciones Distribuidas

La diferencia reside en que el software o la computadora del cliente accede a los datos desde el servidor o cloud, mientras que el servidor o cloud se encarga de procesar los datos.

Si un componente de aplicación distribuida se cae, puede conmutar por error a otro componente para continuar ejecutándose.

Las aplicaciones distribuidas permiten que múltiples usuarios accedan a las aplicaciones a la vez.

# Aplicaciones Distribuidas

Muchos desarrolladores, profesionales de TI o empresas eligen almacenar aplicaciones distribuidas en la nube debido a la elasticidad y escalabilidad de la nube , así como a su capacidad para manejar grandes aplicaciones o cargas de trabajo.

Las empresas pueden optar por utilizar tecnología de contenedores, como Docker , para empaquetar e implementar aplicaciones distribuidas. Los contenedores pueden construir y ejecutar aplicaciones distribuidas, así como aplicaciones distribuidas separadas de otras aplicaciones en una nube o infraestructura compartida.

# Aplicaciones Distribuidas

Existen seis características principales responsables de la utilidad de los sistemas distribuidos:

1. Compartición de recursos.
2. Apertura (openness).
3. Concurrencia.
4. Escalabilidad
5. Tolerancia de Fallos.
6. Transparencia.



# Aplicaciones Distribuidas - Compartir recursos

Un sistema distribuido permite compartir recursos hardware y software (discos, impresoras, ficheros y compiladores) que se asocian con Computadores de una red.

El término recurso es el que mejor abarca toda la variedad de entidades que pueden compartirse en un sistema distribuido incluyendo componentes hardware como discos e impresoras hasta elementos software como ficheros, ventanas, bases de datos y otros objetos de datos.

La idea de compartición de recursos no es nueva ni aparece en el marco de los sistemas distribuidos.



# Aplicaciones Distribuidas - Compartir recursos

Los sistemas multiusuario clásicos prevén compartir recursos entre sus usuarios aunque esta acción se hace de manera natural entre todos sus usuarios.

Los usuarios de estaciones de trabajo monousuario o computadoras personales dentro de un sistema distribuido no obtienen automáticamente los beneficios de la compartición de recursos.

Los recursos en un sistema distribuido están físicamente encapsulados en una de los Computadores y sólo pueden ser accedidos por otros equipos mediante las comunicaciones en red.

Para que la compartición de recursos sea efectiva debe ser manejada por un programa que ofrezca un interfaz de comunicación permitiendo que el recurso sea accedido, manipulado y actualizado de una manera fiable y consistente. Todo esto lleva a la definición de gestor de recursos.

# Aplicaciones Distribuidas - Apertura

Los sistemas distribuidos son normalmente sistemas abiertos lo que significa que se diseñan sistema utilizando protocolos estándar que permiten combinar hardware y software de diferentes fabricantes.

Un sistema informático es abierto si el sistema puede ser extendido de diversas maneras. Un sistema puede ser abierto o cerrado con respecto a extensiones hardware (añadir periféricos, memoria o interfaces de comunicación, etc.) o con respecto a las extensiones software (añadir características al sistema operativo, protocolos de comunicación y servicios de compartición de recursos, etc.).



# Aplicaciones Distribuidas - Apertura

Básicamente los sistemas distribuidos deben cumplir una serie de características:

1. Los interfaces software clave del sistema están claramente especificados y se ponen a disposición de los desarrolladores. Los interfaces se hacen públicos.
2. Los sistemas distribuidos abiertos se basan en la provisión de un mecanismo uniforme de comunicación entre procesos e interfaces publicados para acceder a recursos compartidos.
3. Los sistemas distribuidos abiertos pueden construirse a partir de hardware y software heterogéneo, posiblemente proveniente de vendedores diferentes.

La conformidad de cada componente con el estándar publicado debe ser cuidadosamente comprobada y certificada si se quiere evitar tener problemas de integración.

# Aplicaciones Distribuidas - Concurrency

Estos procesos pueden (aunque no es obligatorio) comunicarse con otros durante su funcionamiento normal. Cuando existen varios procesos en una única máquina decimos que se están ejecutando concurrentemente.

Si la computadora está equipada con un único procesador central la concurrencia tiene lugar entrelazando la ejecución de los distintos procesos. Si la computadora tiene  $N$  procesadores entonces se pueden estar ejecutando estrictamente a la vez hasta  $N$  procesos.

En los sistemas distribuidos hay muchas maquinas, cada una con uno o más procesadores centrales. Es decir, si hay  $M$  Computadores en un sistema distribuido con un procesador central cada una entonces hasta  $M$  procesos estar ejecutándose en paralelo.

# Aplicaciones Distribuidas - Concurrency

En un sistema distribuido que está basado en el modelo de compartición de recursos, la posibilidad de ejecución paralela ocurre por dos razones:

1. Muchos usuarios interactúan simultáneamente con programas de aplicación: Este caso es el menos conflictivo porque normalmente las aplicaciones de interacción se ejecutan aisladamente en la estación de trabajo del usuario y no entran en conflicto con las aplicaciones ejecutadas en las estaciones de trabajo de otros usuarios.
2. Muchos procesos servidores se ejecutan concurrentemente, cada uno respondiendo a diferentes peticiones de los procesos clientes: Esta situación se produce por la existencia de uno o más servidores para cada tipo de recurso. Estos procesos servidores se ejecutan en distintas máquinas, de manera que se están ejecutando en paralelo diversos servidores, junto con diversos programas de aplicación .

# Aplicaciones Distribuidas - Concurrency

Las peticiones para acceder a los recursos de un servidor dado son puestas en las colas de trabajo del servidor y son procesadas secuencialmente o procesadas concurrentemente por múltiples instancias del gestor de recursos.

Cuando esto ocurre los procesos servidores deben sincronizar sus acciones para asegurarse de que no existen conflictos.

La sincronización debe ser cuidadosamente planeada para asegurar que no se pierden los beneficios de la concurrencia.

# Aplicaciones Distribuidas - Escalabilidad

Los sistemas distribuidos son escalables en tanto que la capacidad del sistema puede incrementarse añadiendo nuevos recursos para cubrir nuevas demandas sobre el sistema.

Los sistemas distribuidos operan de manera efectiva y eficiente a muchas escalas diferentes. La escala más pequeña consiste en dos estaciones de trabajo y un servidor de archivos, mientras que un sistema distribuido construido alrededor de una red de área local simple podría contener varios cientos de estaciones de trabajo, varios servidores de archivos, servidores de impresión y otros servidores de propósito específico.

# Aplicaciones Distribuidas - Escalabilidad

A menudo se conectan varias redes de área local para formar internetworks, y éstas podrían contener muchos miles de Computadores que forman un único sistema distribuido, permitiendo que los recursos sean compartidos entre todos ellos.

Tanto el software de sistema como el de aplicación no deberían cambiar cuando la escala del sistema se incrementa.

La necesidad de escalabilidad no es solo un problema de prestaciones de red o de hardwares que está íntimamente ligada con todos los aspectos del diseño de los sistemas distribuidos.

# Aplicaciones Distribuidas - Escalabilidad

El diseño del sistema debe reconocer explícitamente la necesidad de escalabilidad o de lo contrario aparecerán serias limitaciones. La demanda de escalabilidad en los sistemas distribuidos ha conducido a una filosofía de diseño en que cualquier recurso simple -hardware o software- puede extenderse para proporcionar servicio a tantos usuarios como se quiera. Si la demanda de un recurso crece, debería ser posible extender el sistema para darla servicio.

El trabajo necesario para procesar una petición simple para acceder a un recurso compartido debería ser prácticamente independiente del tamaño de la red. Las técnicas necesarias para conseguir estos objetivos incluyen el uso de datos replicados, la técnica asociada de caching, y el uso de múltiples servidores para manejar ciertas tareas, aprovechando la concurrencia para permitir una mayor productividad.

# Aplicaciones Distribuidas - Tolerancia a fallos

Los sistemas distribuidos pueden ser tolerantes a algunos fallos de funcionamiento del hardware y del software.

En la mayoría de los sistemas distribuidos se puede proporcionar un servicio degradado cuando ocurren fallos de funcionamiento. Realmente una completa pérdida de servicio sólo ocurre cuando existe un fallo de funcionamiento en la red.

Los sistemas informáticos a veces fallan. Cuando se producen fallos en el software o en el hardware, los programas podrían producir resultados incorrectos o podrían pararse antes de terminar la operación que estaban realizando.



# Aplicaciones Distribuidas - Tolerancia a fallos

El diseño de sistemas tolerantes a fallos se basa en dos soluciones:

1. Redundancia hardware: uso de componentes redundantes.
2. Recuperación del software: diseño de programas que sean capaces de recuperarse de los fallos.

En los sistemas distribuidos pueden replicarse los servidores individuales que son esenciales para la operación continuada de aplicaciones críticas.

# Aplicaciones Distribuidas - Tolerancia a fallos

La recuperación del software tiene relación con el diseño de software que sea capaz de recuperar (roll-back) el estado de los datos permanentes antes de que se produjera el fallo.

Los sistemas distribuidos también proveen un alto grado de disponibilidad en la vertiente de fallos hardware.

La disponibilidad de un sistema es una medida de la proporción de tiempo que está disponible para su uso.



# Aplicaciones Distribuidas - Transparencia

La transparencia se define como ocultar al usuario y al programador de aplicaciones de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un todo, en vez de una colección de componentes independientes.

La transparencia ejerce una gran influencia en el diseño del software de sistema. Estas proveen un resumen útil de la motivación y metas de los sistemas distribuidos.



# Aplicaciones Distribuidas - Transparencia

Los tipos de transparencias definidas son:

1. **Transparencia de Acceso:** Permite el acceso a los objetos de información remotos de la misma forma que a los objetos de información locales.
2. **Transparencia de Localización:** Permite el acceso a los objetos de información sin conocimiento de su localización
3. **Transparencia de Concurrencia:** Permite que varios procesos operen concurrentemente utilizando objetos de información compartidos y de forma que no exista interferencia entre ellos.



# Aplicaciones Distribuidas - Transparencia

**4. Transparencia de Replicación:** Permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación tengan que conocer la existencia de las réplicas.

**5. Transparencia de Fallos:** Permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.

**6. Transparencia de Migración:** Permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.



# Aplicaciones Distribuidas - Transparencia

**7. Transparencia de Prestaciones:** Permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varía.

**8. Transparencia de Escalado:** Permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

Las dos más importantes son las transparencias de acceso y de localización; su presencia o ausencia afecta fuertemente a la utilización de los recursos distribuidos.

A menudo se las denomina a ambas transparencias de red. La transparencia de red provee un grado similar de anonimato en los recursos al que se encuentra en los sistemas centralizados.

# Aplicaciones Distribuidas - Seguridad

Puede accederse al sistema desde varias computadoras diferentes, y el tráfico en la red puede estar sujeto a hackers indeseables..

Esto hace más difícil el asegurar que la integridad de los datos en el sistema se mantenga y que los servicios del sistema no se degraden por ataques de denegación de servicio.

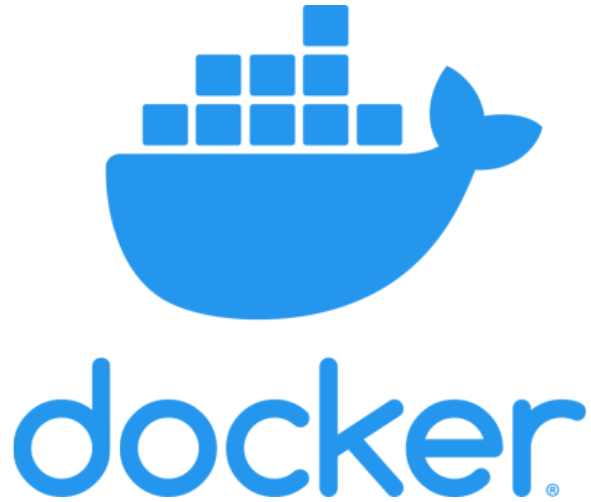
# Aplicaciones Distribuidas - Manejabilidad

Las computadoras en un sistema pueden ser de diferentes tipos y pueden ejecutar versiones diferentes de sistemas operativos.

Los defectos en una máquina pueden propagarse a otras máquinas con consecuencias inesperadas.

Esto significa que se requiere más esfuerzo para gestionar y mantener el funcionamiento del sistema.





# Ejercicio

Instale Alguna herramienta para  
monitorear contenedores de  
Docker



# **Introducción a la Integración de Software**

**Aplicaciones Integradas**



# Aplicaciones Integradas

Los sistemas de aplicaciones integrados son sistemas de infraestructura pre-integrados con bases de datos, software de aplicaciones o ambos, que proporcionan una funcionalidad similar a un dispositivo para soluciones de Big Data, plataformas analíticas o demandas similares.



# Aplicaciones Integradas

Existen 4 razones clave para considerar la entrega de sus soluciones en forma de sistemas de aplicaciones integradas:

**Mayor coherencia de la solución:** en un sistema de aplicación integrado, integra el hardware y el software para su producto, de modo que controla el entorno que lo respalda. Eso garantiza que sus aplicaciones y análisis tengan todos los recursos de procesamiento, almacenamiento y memoria que necesitan para ofrecer un rendimiento óptimo en trabajos de computación intensiva. En resumen, su aplicación se ejecuta de la forma en que fue diseñada, por lo que se optimiza la satisfacción del cliente.



# Aplicaciones Integradas

**Mejor seguridad del sistema:** los análisis y otros sistemas, con frecuencia manejan información financiera, médica u otra información patentada. Al entregar su producto como un sistema de aplicación integrado, puede incorporar las herramientas de seguridad necesarias para evitar el acceso de usuarios no autorizados, piratas informáticos u otros intrusos. Su aplicación es más segura, por lo que sus clientes ganan confianza en sus productos.



# Aplicaciones Integradas

**Ciclo más rápido desde el desarrollo de la solución hasta la implementación:** con una aplicación tradicional, entregada solo como software, sus ingenieros pasan mucho tiempo considerando posibles escenarios de implementación del sistema de análisis y descubriendo cómo maximizar el rendimiento del sistema en cada uno. Eso puede retrasar que su producto llegue a los clientes por meses o más. Pero con un sistema de aplicación integrado, puede especificar el entorno una vez y llevar su producto al campo sin demoras innecesarias.



# Aplicaciones Integradas

**Reduce la carga de soporte para la solución:** un beneficio adicional de la coherencia de la solución es la facilidad de soporte para sus soluciones y análisis. Cuando vende software solo, su personal de soporte nunca puede estar seguro exactamente de qué hardware se está ejecutando, qué más se está ejecutando en la plataforma y una gran cantidad de otras variables. Con un sistema de aplicación integrado, toda esa información está bajo su control, por lo que replicar y resolver problemas se vuelve simple y directo.



# Ejercicio

Realice una tabla de comparación entre aplicaciones distribuidas y integradas