

Softtek®

vector ITC
a Softtek Company

Next-Gen Software: Languages & Tools

Índice

01 **04 - 12**
Lenguajes de Programación

02 **13 - 18**
Lenguajes Orientados a IA y ML

03 **19 - 35**
Frameworks Web

04 **36 - 44**
Movilidad

05 **45 - 52**
Base de Datos

06 **53 - 58**
Data Platform

07 **59 - 65**
Data-Intensive Computation

08 **66 - 74**
Streaming en Tiempo Real

09 **75 - 81**
Motores de Búsqueda y Visualización

10 **82 - 89**
Gestión de Eventos

11 **90 - 95**
Contenedores

12 **96 - 102**
Modelos de Diseño de APIs

13 **103 - 108**
DL Library



01. Lenguajes de Programación

La lista de lenguajes y protocolos de programación existentes en el mercado es cada vez mayor, lo que dificulta la elección del lenguaje más adecuado para un aplicativo específico. Algunos de los lenguajes que están surgiendo son especialmente modernos y simples, pero son capaces de competir con los ya establecidos en el sector.

Existe otro factor a tener en cuenta: la industria del desarrollo de software está en auge y un enorme porcentaje de nuevos desarrolladores se incorporan al sector, por lo que los lenguajes de programación más sencillos y fáciles de desarrollar tenderán a ser los más populares, haciendo que los tradicionales pierdan cuota de mercado. Estos últimos, que cada vez añaden más complementos y bibliotecas, han dificultado enormemente el aprendizaje, generando barreras de entrada que plantean una oportunidad de mercado para los nuevos lenguajes.



LENGUAJES MAINSTREAM



Python y JavaScript dominan el mercado de los lenguajes de programación desde hace tiempo, y siguen ganando cada vez más popularidad.

Python es uno de los lenguajes de programación más conocido y empleado por los desarrolladores. Se caracteriza por ser un lenguaje fácil de aprender, con una estructura limpia y clara, capaz de construir aplicaciones web estables. Es el lenguaje número uno en diversos campos, liderando el sector Data Science.

Se caracteriza por la **amplia gama de bibliotecas que proporciona, con módulos de código abierto que facilitan el desarrollo de aplicaciones**. Se utiliza para aplicaciones web y de escritorio generalmente, pero tiene un gran número de usos: por ejemplo, Python ofrece Django y Flask, bibliotecas populares para desarrollo web y TensorFlow, Keras, SciPy y Pandas para aplicaciones de Data Science.

Los frontales han ido adquiriendo cada vez más responsabilidad desde hace años y han pasado de ser maquetas acopladas al back a aplicaciones web desacopladas cada vez más sofisticadas. Sobre todo desde que desapareció Action Script (Flash), el artífice de esta revolución ha sido JavaScript, un lenguaje caracterizado por su **gran flexibilidad y eficacia**. De hecho, JavaScript se utiliza ya en otros ámbitos además de los frontales web y también se puede usar para el lado del servidor, para preparar aplicaciones web, aplicaciones de escritorio, juegos y hasta entornos de IA.

Además del lenguaje en sí mismo, también destaca la evolución de los grandes frameworks con los que se trabaja en javaScript, como Angular, React o Vue en las aplicaciones web; NestJS (Node) para el lado del servidor; y React Native e Ionic para aplicaciones móviles.

Mención aparte merece el **metalenguaje TypeScript**, una capa que se puede incorporar a JavaScript para incorporar algunas características que no trae este último lenguaje de fábrica, como el tipado (sencillo y complejo) de sus distintas piezas –variables, funciones, clases, etc.-, lo que permite una programación mucho más segura y robusta.

Java se emplea para crear programas y plataformas altamente funcionales. Este lenguaje de programación orientado a objetos no requiere una infraestructura de hardware específica, es fácilmente manejable y seguro.

Dada su antigüedad en la industria, Java es increíblemente estable, aunque en los últimos tiempos se está quedando algo anticuado, y por ello, **está perdiendo lentamente su popularidad como el lenguaje de programación líder**. Sin embargo, con su legendaria compatibilidad y la aceptación de cambios (ciclo de lanzamiento de 6 meses, máquina virtual políglota GraalVM), seguirá siendo la opción número uno en el desarrollo de software empresarial.



A pesar de que C++ es un lenguaje más complejo de aprender que los anteriores mencionados, sus características únicas lo enmarcan entre los grandes lenguajes de programación. Grandes compañías como Microsoft o Adobe han visto su nacimiento gracias a este lenguaje. Presenta un rendimiento muy superior a otros lenguajes, y principalmente se enfoca en el desarrollo de juegos, programas complejos, navegadores (como Google Chrome) o sistemas operativos. En los últimos tiempos, ha experimentado también un gran crecimiento dentro de la RV y RA.

El año pasado se lanzó C++20, que cambiará completamente la forma en que C++ era programado hasta ahora. Es uno de los lanzamientos más esperados de los últimos tiempos, y puede impactar decisivamente en su popularidad.

Scala

A pesar de que Scala nació en 2004, ha cobrado mayor atención en los últimos años. Este lenguaje de código abierto facilita la combinación de codificación funcional y orientación a objetos en un entorno de programación de tipo estático.

Su origen está en la búsqueda por mejorar las carencias de Java y crear un lenguaje escalable. Es por ello por lo que es simple, lógico y organizado, reduciendo la cantidad de código necesario para programar, lo que aumenta la productividad al reducir errores, así como la calidad.

Su sistema de tipado estático fuerza que las abstracciones se utilicen de manera segura y coherente. Por otro lado, el código fuente se compila a bytecode Java para posteriormente ejecutarse en la JVM. Esto permite utilizar características o librerías Java y hace que sea una buena elección para la creación y desarrollo de softwares que se ejecuten en múltiples plataformas.

Sin embargo, su velocidad y capacidad de compilación es peor que Kotlin o Java. Además, es un lenguaje muy difícil de comprender por lo que presenta dificultades en el aprendizaje inicial de la programación para desarrolladores junior.



LOS LENGUAJES MODERNOS GANAN TERRENO

Los lenguajes más antiguos tienen el bagaje del pasado y no pueden avanzar rápido (Java) o se vuelven aún más pesados e intimidantes al añadir más y más características (C++). Esto no ocurre con los lenguajes más nuevos que pueden satisfacer la demanda de los desarrolladores de hoy en día.

La mayoría de ellos se centran en la ergonomía del desarrollador y admiten características modernas como la inferencia de tipos, seguridad nula, expresividad o brevedad. Muchos de ellos también están diseñados para aprovechar el hardware y la infraestructura modernos (Cloud, Multi-Core, GPU, TPU).

Rust, Kotlin, Go y Swift son algunos de los lenguages que más popularidad están ganando:



Rust es un lenguaje **cuyo objetivo es implementar aplicaciones o sistemas operativos relacionados con Windows, Linux o macOS**. Es un proyecto de código abierto que permite también programar aplicaciones web a menor escala que lenguajes tradicionales. Es un lenguaje de bajo nivel, muy abstracto, cuyo primer caso de uso fue el desarrollo del navegador Firefox. De hecho, fue desarrollado por la Mozilla Foundation, pero a principios de año se traspasó el control y la marca a la Rust Foundation por que el proyecto estaba creciendo más allá de lo que Mozilla podía soportar.

Una característica fundamental de este lenguaje es que evita que se produzcan daños en los códigos si tiene lugar un error, es decir, **aporta seguridad y fiabilidad en el ámbito de la gestión de errores, ya que garantiza la protección de la memoria**. En este sentido, el compilador Rust comprueba si puede haber errores en la memoria durante el tiempo de compilación.

Rust es un lenguaje sencillo de aprender ya que, además de aportar recomendaciones cuando detecta errores, no difiere en demasía de los más tradicionales.

Durante los últimos cinco años, **es el lenguaje de programación más valorado** según la encuesta Stack Overflow Developer Survey. Los grandes de la tecnología están utilizando Rust en lugar de C/C++ para proyectos complejos y sensibles a la seguridad. Tanto es así que Microsoft declaró recientemente que usarían Rust para arreglar los fallos de seguridad, Amazon declaró que también invertiría en el, y Google ya desarrolla Android Open Source Project sobre Rust, e incluso ya hay planes cercanos de que sustituya a C en el Kernel de Linux.

Kotlin

Kotlin es un lenguaje pragmático, orientado principalmente a objetos, pensado para funcionar con la Máquina Virtual de Java (JVM) y Android. Es open source, sencillo y fácil de usar y destaca por su seguridad y claridad. Además, es capaz de trabajar con Java sin requerir capas de adaptación siendo compatible con él 100%, e incluso se está empezando a utilizar también en aplicaciones backend (por ejemplo, está dando compatibilidad con spring y spring boot).

Adicionalmente, **permite reducir las líneas de código, aproximadamente en un 40% con respecto a otros lenguajes, ahorrando recursos y costes**. Esto presenta mayor velocidad de desarrollo, es decir, con un menor número de líneas de código se pueden realizar gran cantidad de funcionalidades y así, se acelera el proceso de desarrollo de una aplicación y mejora el rendimiento en su producción.

Aunque la gran ventaja de Kotlin son las corrutinas, que permiten manejar hilos en segundo plano de una manera muy sencilla y mucho más potente que de las maneras tradicionales, aumentando la productividad de los desarrolladores.

Por otro lado, cuenta con un sistema integrado a prueba de fallos que, en consecuencia, reduce drásticamente el riesgo de errores en la codificación realizada por el programador.

Es el segundo lenguaje de programación más popular en el ecosistema de JVM, y en años anteriores Google lo ha declarado el lenguaje preferido para los desarrolladores de aplicaciones de Android. También, se emplea para el desarrollo de aplicaciones web, de escritorio y de Server-Side.



Golang, también conocido como Go, es el nuevo lenguaje de programación open source creado por Google. **Se caracteriza por ser simple, con un vocabulario muy sencillo y una curva de aprendizaje rápida y fácil**. Al estar basado en C, tiene muchas características similares con este, así como con Python, lo que resulta muy práctico para desarrolladores con conocimientos previos en lenguajes más clásicos. Se emplea principalmente para servidores web, canalizaciones de datos y aplicaciones de aprendizaje automático.

Se trata de un lenguaje muy potente con un rendimiento elevadísimo, ya que su compilador pasa directamente el código escrito a código máquina, siendo totalmente orientado a objetos. Además, posee un recolector de basura, funciones de reflexión y compilación cruzada. Con muchas mejoras programadas para el lanzamiento de Go 2, ganará aún más popularidad en el futuro.

Este lenguaje cuenta con **Goroutines**. Estas no son más que funciones básicas que pueden ejecutarse de forma independiente, brindando la capacidad de ejecutar varios procesos simultáneamente cuando es necesario, sin apenas uso de memoria RAM, reduciendo así el riesgo de fallo por falta de memoria.

Sin embargo, **aunque la sintaxis es más sencilla, al estar basada en C presenta una codificación prolongada**, por lo que puede necesitar escribir muchas líneas de código que con otros lenguajes. Además, como no cuenta con funciones genéricas se reduce la eficiencia durante el desarrollo debido a la falta de reutilización del código.





Swift

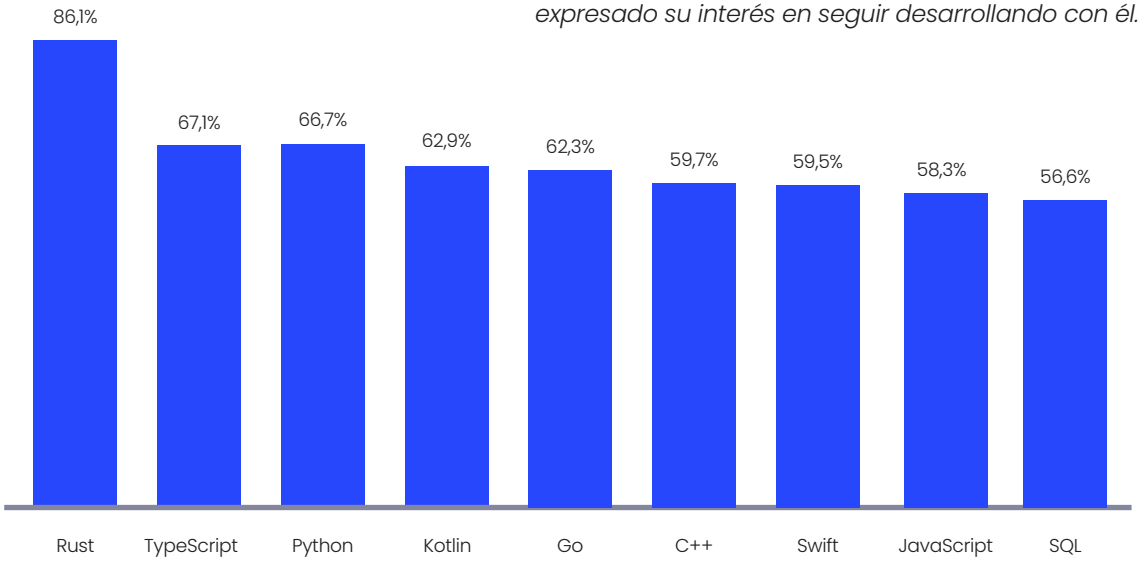
Swift fue creado en 2014 por Apple para las aplicaciones de IPhone e IPad, y actualmente ha llegado a reemplazar a los lenguajes basados en C (C, C++ y Objective-C) como el lenguaje principal de Mac, IOS, Apple TV y Apple Watch.

Surgió ante la demanda de un lenguaje que redujera errores (advierte de ellos antes de que el código se compile) y que tuviese una sintaxis concisa. Este lenguaje de código abierto se caracteriza por ser simple, versátil y con aplicaciones prácticas. Además, es rápido, seguro y fácil de aprender.

Lo que aporta frente a Objective-C es la modernidad, simplicidad y la reducción de bloqueos en las aplicaciones. Además, en caso de que sea una aplicación previamente construida con Objective-C, Swift puede combinarse con él.

Lenguajes de programación más populares para 2021

Datos: Porcentaje de desarrolladores que están desarrollando con el lenguaje o la tecnología y han expresado su interés en seguir desarrollando con él.



Stack Overflow Annual Developer Survey.



02. Lenguajes Orientados a IA

Debido a que **Python permite crear algoritmos que procesen la información de forma inteligente, es un lenguaje muy útil para las diferentes tecnologías basadas en IA**. La previsión para este lenguaje de programación es que aumente notablemente su uso a medida que crecen el Machine Learning y la Inteligencia Artificial, como ha venido ocurriendo hasta ahora.

Dispone de una amplia gama de bibliotecas para el ámbito de la IA, capaces de controlar y transformar datos para el ML que requiere un procesamiento continuo de los mismos.

Algunas de las bibliotecas más usadas de Python son:



INTELIGENCIA ARTIFICIAL

SimpleAI **EasyAI** **PyDatalog**



MACHINE LEARNING

Scikit-Learn **PyBrain** **PyML**



DEEP LEARNING

Keras

Para cálculos rápidos y funciones de creación de prototipos.

TensorFlow

Para redes neuronales con grandes conjuntos de datos.



VISIÓN ARTIFICIAL

OpenCV

Biblioteca libre desarrollada por Intel con aplicaciones como reconocimiento de objetivos, detección de movimiento, etc. Aunque está desarrollada con C y C++ es perfectamente compatible con Python, por lo que se facilita su uso.



NLP

NLTK

Implementa documentación para la investigación del lenguaje natural, datos lingüísticos y habilita el análisis de texto para diferentes plataformas.

En general, Python es un lenguaje muy fácil de aprender y permite trabajar cómodamente con sistemas complejos. También, existe la posibilidad de transferir el proceso de una plataforma a otra con el código ejecutable requerido para cada una de ellas. Es, por tanto, **un lenguaje muy versátil, capaz de ejecutarse en diferentes sistemas** como Windows, MacOS o Linux. Una vez más, esta cuestión ahorra tiempo y dinero permitiendo que el proceso general sea más simple.

Por otro lado, Python **es flexible**, porque permite seleccionar el estilo de programación según la aplicación que necesita ser desarrollada, se puede combinar sin esfuerzo con otros lenguajes de programación y tampoco requiere que se vuelva a compilar el código fuente.

Por último, es muy útil el **fuerte apoyo de la comunidad** formada alrededor de este lenguaje de programación. Siendo un lenguaje de código abierto, permite que haya una gran cantidad de recursos existentes para los programadores. El lenguaje de programación Python es absolutamente gratuito, al igual que la variedad de sus bibliotecas y herramientas.



CASOS DE USO DE PYTHON PARA IA

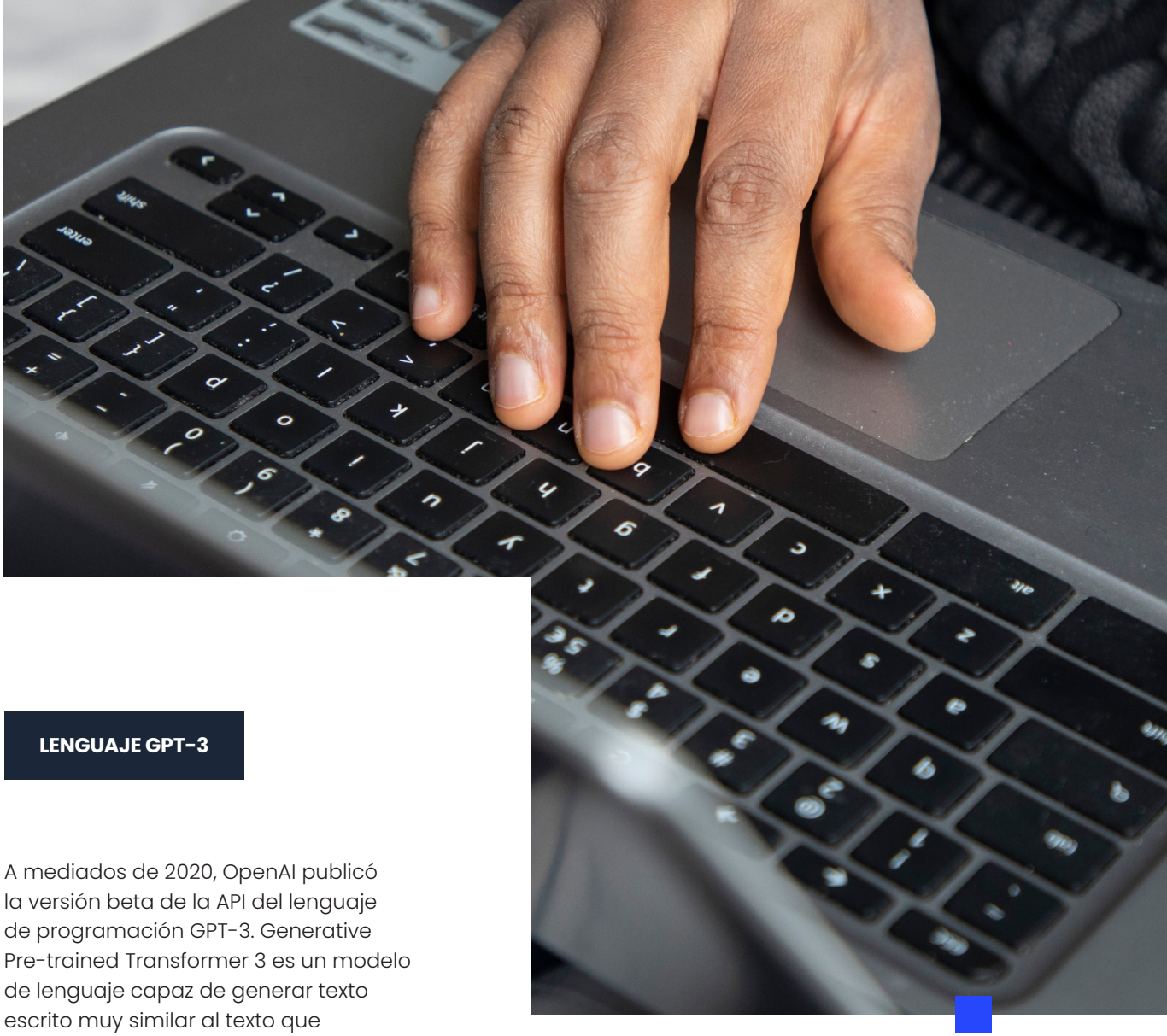
Python ofrece muchas características que son útiles para la IA y el ML. No es de extrañar que varias industrias lo utilicen para predicciones y aprendizaje automático.

Este es el caso de **SkyScanner** que mediante Python desarrollo un algoritmo de ML para sugerir destinos a los usuarios, o **Uber**, que desarrolló una plataforma de ML llamada Michelangelo PyML.

También, se utiliza para construir soluciones como **Anaconda** para analizar eficazmente el mercado, hacer predicciones y visualizar datos en relación con las criptomonedas.

Actualmente, existen diferentes servicios basados en Python, como softwares de banca online que permiten a los usuarios realizar y controlar sus pagos, o crear una red social dentro del software, para que las personas puedan mantenerse conectadas.

Skyscanner Uber ANACONDA



LENGUAJE GPT-3

A mediados de 2020, OpenAI publicó la versión beta de la API del lenguaje de programación GPT-3. Generative Pre-trained Transformer 3 es un modelo de lenguaje capaz de generar texto escrito muy similar al texto que escribiría una persona.

Este lenguaje es una forma de Inteligencia Artificial, basado en Machine Learning, donde los algoritmos son capaces de reconocer patrones en los datos y aprender a partir de ejemplos, formando una red neuronal recurrente con memoria.

La idea de funcionamiento del modelo es sencilla: se analiza el texto y los datos involucrados en la conversación y, a continuación, ofrece predicciones en base a la información previa. Llegado a este punto, la IA **desarrolla por sí misma las respuestas y frases a las peticiones que se le introducen**.

OpenAI ha desarrollado este sistema inteligente gracias al procesamiento de datos de una gran cantidad de texto de Internet, incluida la Wikipedia. En consecuencia, el modelo ha aprendido a escribir en inglés y a simular la forma de expresarse que tiene el humano en internet.

Actualmente, **la forma de entrenamiento empleada para GPT-3 es la conocida como autosupervisión**, es decir, los algoritmos predicen la siguiente palabra de los textos escritos por una persona; en caso de ser correctos continúan la predicción y, en caso de fallo, se corrigen según el texto original, entrenándose a sí mismos.

¿PARA QUÉ SE UTILIZA?

Los dos principales usos que se esperan de GPT-3 son la capacidad de completar textos y su capacidad de programar en cualquier lenguaje. Respecto a la tarea de completar textos, es capaz de hacerlo con soltura a partir de una frase inicial descrita por una persona, adaptándolo a diferentes estilos de escritura. Por ejemplo, se puede emplear una jerga más formal si se busca el desarrollo de texto para un juicio o de artículo en caso de que sea un texto para un periódico.

Paralelamente, su aplicación más relevante y sencilla es su uso como buscador de internet, al igual que Google, con la diferencia de que Google únicamente realiza su búsqueda a través de palabras clave mientras que **GPT-3 basa su búsqueda en la semántica del lenguaje natural junto con palabras clave**.



Aunque el mejor funcionamiento para el modelo es el inglés, GPT-3 conoce diferentes idiomas ya que al recolectar contenido de internet ha obtenido páginas traducidas o directamente escritas en otros idiomas.

Asimismo, el uso principal que se le está dando es la programación, ya que es capaz de generar un código en Python a través de una breve indicación. Esta aplicación es una de las más interesantes, pero aún se está descubriendo. Sin embargo, los avances que promete son inmensos.

Sin embargo, GPT-3 todavía no es una solución práctica para algunos problemas industriales y, actualmente, **su uso se emplea para investigación y desarrollo**. El modelo aún presenta limitaciones y unos altos costes computacionales, a lo que se debe sumar, el obstáculo de los sesgos en la mayoría de los algoritmos de aprendizaje automático.

CONCLUSIÓN

El ML y la IA son las tecnologías transversales de más rápido crecimiento en la actualidad. Gracias a su simplicidad, **Python está ganando impulso en este ámbito debido a su robusto y completo conjunto de características**.

Este lenguaje de programación se puede usar en multitud de sectores, desde los viajes a la medicina, pasando por la banca o los seguros. Siendo adecuado para diversos sistemas operativos, disminuye la posibilidad de errores, y por ello es de los lenguajes de programación favoritos en el mercado.

Analizando los nuevos lenguajes orientados a Inteligencia Artificial, **uno de los que más impacto están causando es GPT-3. Aunque aún presenta limitaciones, las implicaciones que conlleva para el avance de la IA son inmensas y suponen una gran revolución**.

Su aplicación se puede observar en diferentes categorías tecnológicas como puede ser el procesamiento del lenguaje humano, la traducción, los asistentes virtuales o la búsqueda. También, ha demostrado tener un importante potencial en el ámbito de la programación.

En conclusión, es una forma de Inteligencia Artificial capaz de generar códigos en lenguaje de programación, así como generación de textos y análisis de estos, todo ello sin representar ningún tipo de inteligencia abstracta.





03.
Frameworks Web

FRAMEWORKS CLIENT-SIDE



Acorde a la complejidad y sofisticación que han ido adquiriendo las aplicaciones web, las herramientas de desarrollo han ido siendo cada vez más variadas y refinadas.

De hecho, el paisaje tecnológico front cambia a tal velocidad que resulta fundamental contar desde el principio con arquitecturas y sistemas muy sólidos que permitan el escalado, la actualización constante y la convivencia de distintas herramientas y versiones de las mismas.



ESTILOS

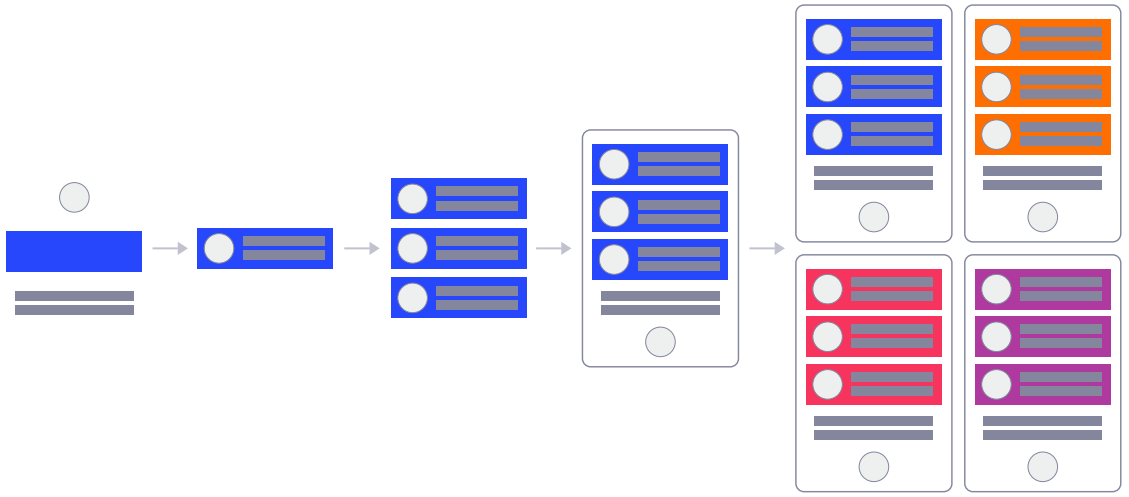
El trabajo con los estilos, la maquetación con CSS, también ha experimentado grandes cambios y hoy en día cuenta con un ecosistema de complejidad creciente.

Entre las herramientas sigue destacando **SASS**, el preprocesador CSS más popular, con cual se pueden tratar los estilos de forma más ergonómica y escalable gracias a las variables, los módulos y las funciones reutilizables entre otras prestaciones.

Igual de popular es el postprocesador **PostCSS**, que permite escribir CSS más legible y ordenado al optimizar el bundle final según las necesidades de los navegadores, además de aportar otra serie de utilidades como la minificación o la ofuscación.

Respecto a la metodología, **BEM** (bloque – elemento – modificador) y sus derivados sigue pareciendo una opción preferible a las OOCSS, dada su perfecta combinación con los sistemas modulares de diseño, como las propuestas del **Atomic Design**.

Esquema conceptual del Atomic Design:



A las librerías más populares, como **Materialize** o Boostrap, también se ha unido recientemente **Tailwind CSS**, aunque los especialistas en maquetación no terminan de verla con buenos ojos por la verbosidad que termina por conferir al HTML.

FRAMEWORKS JS

En general, las web modernas se desarrollan siguiendo un sistema conocido como **Single Page Application (SPA)**, en las que el frontal funciona de forma autónoma, es capaz de gestionar la lógica de negocio y se comunica con el back a través de un API Rest. Consiste, en esencia, en **desarrollar una aplicación web en lugar de una página web**.

Entre sus principales características cabe destacar:

- Óptima experiencia de usuario.
- Descargan operaciones del servidor.
- Alta complejidad técnica.
- Comunicación por API REST.
- Desacopladas del back.
- Testables
- Performance
- Modulares y escalables.
- I18n.
- Gestionan el enrutado.
- Seguridad.



Por su complejidad, **para desarrollar una SPA necesariamente hay que contar con un framework**. Los tres más relevantes que hay ahora en el mercado son Angular, React y Vue.

ANGULAR está desarrollado por Google. Las versiones modernas se caracterizan por la incorporación de la librería rxjs para el manejo de los eventos asíncronos y viene de partida con typescript. Es la herramienta más aconsejable para aplicaciones web de grandes dimensiones, sobre todo si son de naturaleza administrativa, con numerosas operaciones CRUD (altas, bajas, modificaciones).

REACT es un framework de Facebook orientado a la programación funcional. Permite el trabajo con typescript y se ha convertido en el framework más popular del mercado. Entre sus virtudes se encuentran la buena performance y la curva de aprendizaje, mucho menor que Angular. Entre sus defectos, la necesidad de complementar con numerosas librerías auxiliares –como Redux- algunas funcionalidades que en Angular forman parte del propio framework.

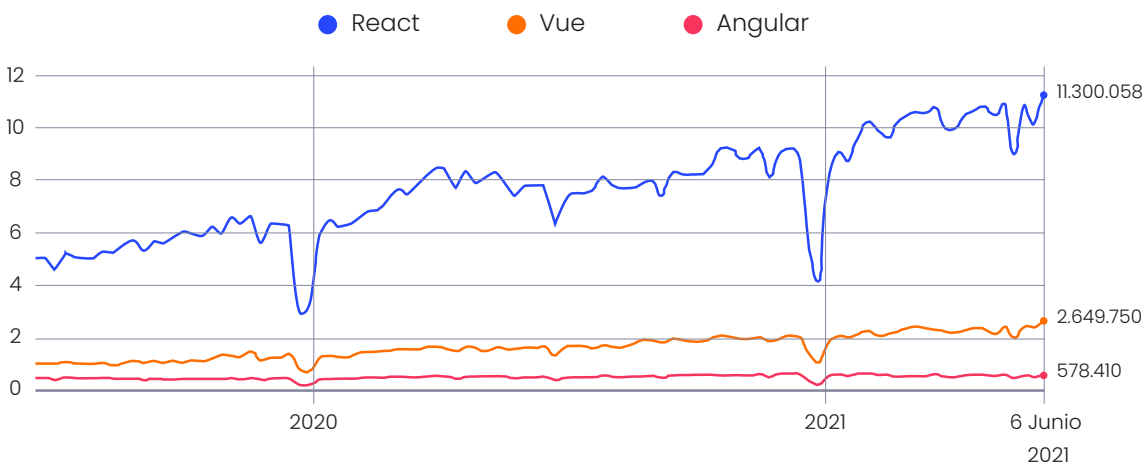
VUE es el más joven de los tres frameworks, pero desde 2016, aproximadamente, ha ido ganando cada vez más popularidad. Combina estrategias de React (Vuex) y Angular. Como en el caso de React, debe complementarse con librerías auxiliares.

La tecnología más adecuada para el desarrollo de un proyecto depende del contexto:

No existe una razón a priori para elegir un framework u otro, todos tienen sus virtudes y sus defectos. **Siempre depende de las necesidades y posibilidades del proyecto.**

	Angular	React	Vue
¿Es una aplicación muy grande?	✓	✗	✗
¿La performance es fundamental?	✗	✓	✓
¿Es un proyecto multidispositivo?	✓	✗	✗
¿El equipo maneja muy bien JavaScript?	✓	✗	✗
¿Necesita articularse en microfrontends?	✓	✗	✓
¿Es un aplicación de gestión?	✓	✗	✓

Descargas de los últimos 2 años de Angular vs React vs Vue:



Fuente: npm trends

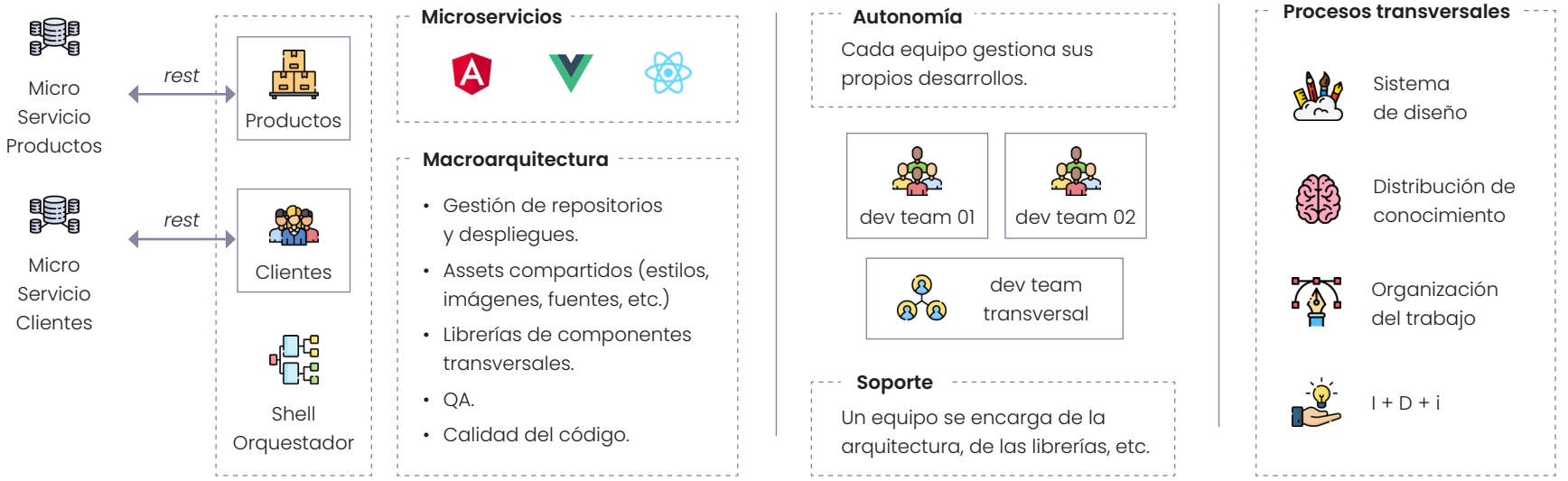


MICROFRONTENDS

Cuando las aplicaciones son muy grandes y cuentan con decenas y decenas de páginas, el modelo de una single page no basta para cubrir todas las necesidades técnicas, sobre todo si hay que integrar aplicaciones antiguas realizadas con distintas tecnologías. **Es un caso habitual en las aplicaciones web relacionadas con la gestión y la administración de una corporación**.

En estos casos, conviene implementar una arquitectura basada en microfrontends, que en esencia consiste en trabajar con las distintas piezas de una aplicación de forma separada y luego estructurarlas en un armazón que se encargue del ruteado, la gestión del estado y demás necesidades transversales.

Esta estrategia, que permite combinar distintas herramientas si fuera necesario, **va más allá del marco tecnológico y conlleva también ciertos planteamientos metodológicos**.



Existen diversas maneras de implementar un sistema de microfrontends: desde fórmulas clásicas (iframes) a técnicas más modernas, como el metaframework Single-Spa, el plugin de webpack Module Federation (que está aún en una fase experimental) o la transformación de las aplicaciones en web components, que es la fórmula más recomendable cuando el stack tecnológico principal se basa en Angular.

PWA

Independientemente de que se esté trabajando con un framework u otro, **las Single Page Applications se complementan con estrategias PWA (Progressive Web Apps)** una serie de propuestas para incorporar funcionalidades propias de las apps móviles a las apps web, como la posibilidad de ser instaladas o funcionar offline. A diferencia de las apps móviles, se encuentran en Internet, no necesitan ser descargadas e instaladas, por lo que aumenta la llamada findability, la facilidad de se encontradas de forma natural.

Sus tres rasgos principales son:



PUSH NOTIFICATIONS: Mensajes emergentes, notificaciones que permiten volver a interactuar de forma automática y entregar contenido nuevo siempre que esté disponible.



SERVICE WORKER: Comandos que se ejecutan en segundo plano, separado de una página web para desarrollar operaciones que no interfieran con la experiencia de usuario y que permiten algunas estrategias avanzadas de cacheo de las llamadas REST.



APPLICATION SHELL ARCHITECTURE: Una propuesta arquitectónica que consiste en separar las templates, la carcasa, del contenido, que siempre se obtiene de forma dinámica y es susceptible de ser cacheado.



PRUEBAS AUTOMATIZADAS EN LOS FRONTALES

Los desarrollos front modernos son muy complicados, por lo que se necesitan sistemas de pruebas automáticas que, cuanto menos, nos **permitan minimizar la aparición de bugs e ir escalando una aplicación con seguridad**.

En esencia, podemos distinguir **dos tipos de pruebas**:



UNITARIAS

Miden las partes de un proceso. Dependen de cómo se hace, no de lo que hace.



END TO END

Miden todo un proceso. Dependen de lo que hace, no de cómo lo hace.

*Las **pruebas unitarias** se caracterizan por lo que agrupa el acrónimo FIRST:*





Fast:
rápidas.



Isolated/Independent: deben ser atómicas, cada test mide una cosa (un método)



Self-Validating: se validan por sí mismas



Repeatable: los mismos datos deben dar los mismos resultados



Timely: se escriben antes del código.

HERRAMIENTA DESTACABLE



Entre las herramientas más punteras para desarrollar pruebas unitarias destaca **Jest**, con la que, además de realizar las pruebas tradicionales, se pueden realizar ***snapshots***, instantáneas del renderizado de una vista que se pueden comparar para ver si se han producido cambios indeseados.

Durante mucho tiempo parecían la opción más interesante para cubrir los frontales, pero en este caso **presentan algunos problemas que no se presentan en el back**. Por ejemplo:

- Pueden obligar a programar de forma incorrecta (Ley de Deméter) al tener que dejar todos los métodos accesibles cuando se fijan porcentajes de cobertura arbitrarios.
- En muchas ocasiones, en realidad lo que se está comprobando es que un framework funciona (se hace click y se dispara alguna acción bindeada) y no si el código desarrollado es correcto.
- Las pruebas solo son comprensibles por programadores.
- Son laboriosos, por lo que tienden a dejar de hacerse en la vida real si los presupuestos son ajustados.
- Si cambia el código del proyecto, por ejemplo al cambiar de framework, los test desarrollados dejan de ser útiles y todo ese esfuerzo e inversión se pierde también.

*Las **pruebas End to End**, por el contrario, no se realizan contra la herramienta, sino contra el producto final. Emulan las acciones del usuario para comprobar los resultados. Dependen en suma de lo que hace una aplicación, no de cómo está hecha.*

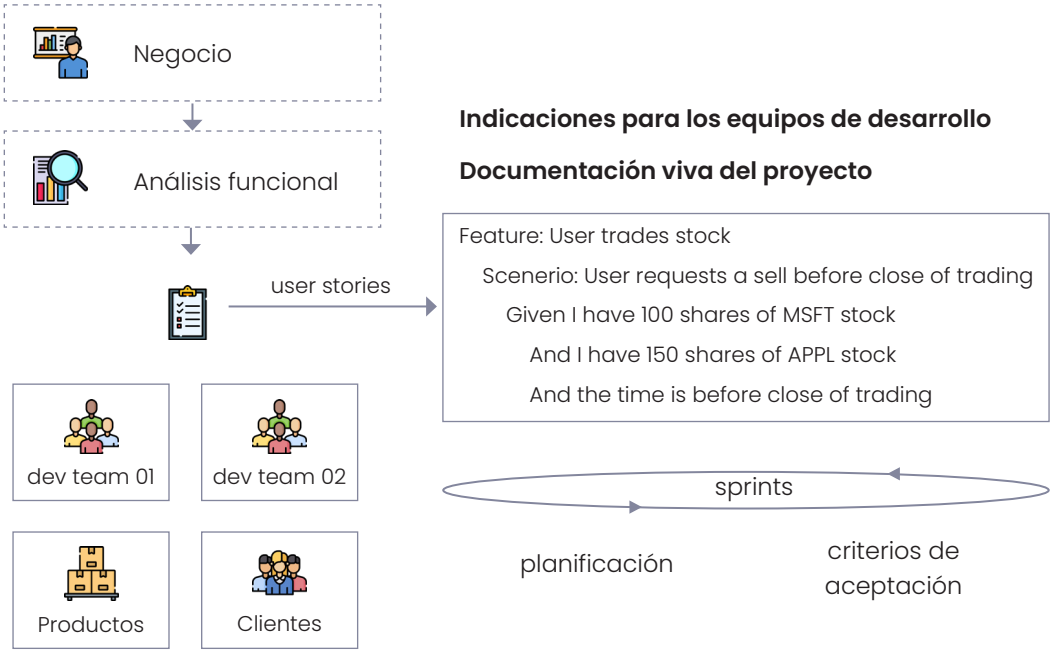


HERRAMIENTA DESTACABLE

cypress

A las herramientas clásicas para hacer este tipo de pruebas, como Selenium, se ha sumado últimamente **Cypress**, un framework moderno muy potente, de fácil configuración, sin dependencias, muy versátil y con una curva de aprendizaje razonable.

Además de mantener la aplicación, sobre todo cuando se combinan con Gherkin, permiten mantener una documentación viva y centralizada de un proyecto, fijar el alcance mediante criterios aceptación y llevar un seguimiento real del estado del desarrollo.



FRAMEWORKS SERVER-SIDE

Los marcos web server-side son framerworks que **facilitan la escritura, el mantenimiento y la escala de aplicaciones web**. Proporcionan herramientas y bibliotecas que simplifican las tareas comunes de desarrollo web, la interacción con bases de datos, las sesiones de soporte y la autorización del usuario, también, mejoran la seguridad frente a los ataques web. Actualmente, sólo se usan para el back-end, pero estos marcos ofrecen un desarrollo de la Web end-to-end, incluyendo capas de vista.



Spring en un framework open source que facilita la construcción de app en Java, Kotlin y Groovy. **Posee un código fuente de los más simples** en este contexto de trabajo y es más sencillo de trabajar que otros marcos. Su función principal es simplificar el desarrollo de las aplicaciones. Aporta todas las características y herramientas necesarias para crear todo tipo de aplicaciones y, además, tiene la capacidad de realizar pruebas unitarias y de integración fácilmente.

Sus características principales pueden ser utilizadas por las aplicaciones de Java y **algunas de sus ventajas** son:



No se necesita configuración para código repetitivo



Fácil gestión de las propiedades específicas del perfil



Personalización facil de las propiedades de la aplicación.

Sigue el paradigma de **Programación Orientada a Aspectos**, permitiendo la implantación de rutinas transversales, y también el patrón MVC.

Mediante una API traduce excepciones específicas de la tecnología en excepciones no verificadas y consistentes. Además, se basa en contenedores **Inversion of Control (IoC)**, que son menos pesados que otros como Enterprise JavaBeans, lo que hace que el framework sea ideal para máquinas con recursos y memoria limitados.

Este framework se caracteriza por que **sus servidores son portátiles e independientes** y, esto permite hacer uso de metaframeworks que incluyan otros componentes externos y otros frameworks.

Por tanto, Spring está diseñado para la **implementación de aplicaciones complejas permitiendo desarrollar soluciones nativas**. Esto le confiere gran prestigio y protagonismo en relación con los frameworks usados para back-end. Así pues, se espera que, en 2021, siga a la cabeza de los marcos server-side.

ASP.NET Core

ASP.NET Core es la nueva estructura más sencilla y modular de ASP.NET, rediseñada completamente. Como este tipo de frameworks, se ejecuta en el servidor web de cualquier explorador. Además, funciona con los marcos de trabajo de JavaScript como React, Angular o Vue.js.

Es open source y gratuito, y su **objetivo es compilar aplicaciones modernas habilitadas para la nube y conectadas a internet**. Se encarga de controlar la apariencia y el funcionamiento de los sitios web, haciendo uso de temas y máscaras, los cuales se definen para posteriormente aplicarse en las páginas o controles.

Es un **framework multiplataforma** permite ejecutarse y desarrollarse en Linux, Mac y Windows y el backend utiliza en todas las plataformas C#. Además, permite utilizar patrones de programación asíncrona.

Cuenta con el **patrón MVC** para realizar las pruebas en las aplicaciones y en las APIs, y posee un control de versiones en paralelo.

Por otro lado, la tecnología de esta solución requiere menos programación, ya que los desarrolladores construyen menos declaraciones. Esto se traduce en una **reducción del mantenimiento necesario**.

Esta nueva versión de ASP.NET dará un nuevo enfoque durante el 2021 para el desarrollo server-side.





django

Django es un framework open-source escrito en Python para el desarrollo web que sigue el diseño de modelo-vista-controlador (MVC), aunque Django lo ha redefinido como **Modelo-Vista-Templete (MVT)**. Se emplea para un rápido desarrollo, un diseño pragmático y la creación de sitios web seguros.

Django busca simplificar el trabajo y optimizar el tiempo de los desarrolladores mediante; la **filosofía DRY**, que se basa en que no es necesario reescribir el código existente porque permite ensamblar el sitio web como si fueran bloques y reutiliza códigos ya existentes; y el **principio de desarrollo rápido**, que significa que los desarrolladores pueden hacer más de una iteración a la vez sin comenzar la programación desde cero.

Por otro lado, Django permite a los desarrolladores usar **diversas bibliotecas**, entre las que destacan: Django REST, cuya función es la construcción de APIs, Django CMS, para administrar el contenido del sitio web, o Django-allauth, que es un conjunto integrado de aplicaciones para el registro y autentificación de cuentas.

Además, destaca su capacidad para abordar **diversos tamaños** de proyectos, es decir, demuestra tener una adaptación versátil a cualquier tipo de proyecto de aplicación web, ya sea un sitio web simple o una aplicación web de alta carga.

Por último, el **ORM** de Django tiene capacidad para extraer información rápidamente, acelerando así el desarrollo de las aplicaciones web, además, ayuda a la construcción de prototipos funcionales, haciendo que no sea necesario conocer la totalidad del lenguaje para la manipulación de datos.

Laravel

Laravel es un framework de PHP de código abierto, simple, potente y con una interfaz versátil. Destaca por el énfasis de la calidad de su código, la poca dificultad de mantenimiento y su escalabilidad, que permite el desarrollo de diferentes tamaños de proyecto. Este framework es más moderno que sus competidores y en los últimos años se ha establecido como **el más utilizado de PHP**, lo que se debe principalmente a su facilidad de uso, ya que su curva de aprendizaje es rápida y sencilla.

En el ámbito de la arquitectura, trabaja con **arquitectura de carpetas avanzada**, es decir, que promueve la separación de archivos en un orden bien definido. También, cuenta con una **arquitectura de clases** que busca la separación del código por responsabilidades.

Laravel cuenta con **Blade, un motor de plantillas** preinstalado para conectar los modelos de datos, procesar el código, dirigir la salida a un archivo de texto o flujo especifico, visualizar los datos y ampliar los diseños sin afectar al rendimiento de la aplicación.

Por otro lado, cabe destacar su propia **interfaz de línea de comandos (CLI) Artistan**, que es capaz de simplificar y acelerar el proceso de desarrollo web. Esta herramienta elimina las tareas de codificación repetitivas y tediosas, genera archivos MVC y administra configuraciones de datos creando comandos propios.

Asimismo, mejora el rendimiento proporcionando una manera sencilla de construir modelos usando **Eloquent ORM**. Una de las mejores herramientas ORM ya que facilita las consultas a bases de datos con una sintaxis PHP simple, sin ser necesario escribir código complejo en SQL y, en consecuencia, ahorra tiempo.

Por último, destacar que permite realizar a las aplicaciones **pruebas unitarias a través de PHPUnit**, capaz de configurar automáticamente los archivos para la realización de las pruebas y además, puede ejecutar varias al mismo tiempo. Esta característica evita errores, hace más eficientes las aplicaciones y las convierte en escalables.

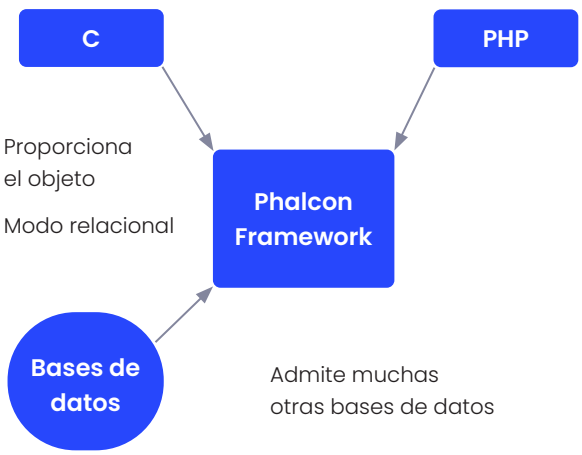


Phalcon es un framework PHP que está basado en la web, de código abierto, con licencia BSD y depende del **patrón MVC**. Gracias a este patrón ofrece diversas características como un ORM, almacenamiento en caché, plantillas y seguridad. Generalmente, se emplea para desarrollar aplicaciones de bajo consumo de energía y alto rendimiento.

Se caracteriza por ser un **marco débilmente acoplado**, es decir, que permite que sus objetos sean tratados como componentes que hacen de pegamento según las necesidades de las aplicaciones. Esto implica que se requiera una menor cantidad de código para los diversos componentes, creando así un marco independiente.

Este framework se ha implementado como una **extensión de servidor web escrita en C y PHP** con el fin de aumentar la velocidad de ejecución y ser capaz de manejar más solicitudes HTTP por segundo que otros marcos. PHP y C pueden usarse como módulos independientes, dando lugar a otro tipo de rendimiento, a la par que es capaz de brindar el mismo soporte que las bases de datos relacionales y no relacionales. Por lo general, es un framework que requiere menos codificación para la implementación de tareas respecto a los demás.

Además, apenas genera una sobrecarga en su aplicación y contiene su propia versión de SQL conocido como PHQL, que se emplea para escribir consultas de bases de datos relacionales para múltiples sistemas de bases de datos.





Ruby on Rails, también conocido simplemente como Rails, es un marco web escrito en lenguaje Ruby con licencia MIT. Está **formado por un conjunto de librerías y automatismos** que buscan solucionar problemas recurrentes en base a la simplicidad, escribiendo menos código y con un mínimo de configuración.



Su filosofía se basa en los principios **DRY**, que busca evitar la duplicidad de código; **Convención sobre Configuración (CoC)**, que asigna parámetros de forma automática sin requerir la intervención de un usuario y sin archivos de configuración; y que es **dogmático**, por lo que asume que hay una forma mejor de hacer las cosas, e incluso desalienta alternativas. Todo ello da como resultado un menor uso de código y aumento de productividad.

Este framework cuenta con **RubyGems**, gestor de paquetes de Ruby, desde donde se distribuye y gestiona la instalación de las bibliotecas y aplicaciones para obtener nuevas funcionalidades sin desarrollarlas desde cero.

La arquitectura sigue el **patrón MVC**, que organiza y separa los códigos en base a las funcionalidades con el objetivo de que los cambios en una parte del código no perjudiquen a otras partes.

Para hacer pruebas automatizadas utiliza **Minitest** por defecto, que acepta mocking, benchmarking, Test Driven Development (TDD) y Behavior Driven Development (BDD). Estos dos últimos enfoques se centran en definir las pruebas antes del desarrollo por lo que permiten reducir costes.

Por último, destacar que muchos desarrolladores lo llaman el "framework ágil por excelencia" debido a que está basado en metodologías ágiles y aumenta la productividad.

CONCLUSIÓN

Mientras que los frameworks client-side simplifican los diseños y las tareas de presentación, los relacionados con server-side proporcionan funcionalidades web comunes. Es decir, la programación server-side **permite distribuir eficientemente la información a medida para cada usuario y, por tanto, mejora la CX**.

Por otro lado, los lenguajes de client-side también se utilizan para crear contenidos dinámicos, pero es el cliente o usuario el que procesa y ejecuta los scripts, de forma que estos scripts se incluyen en el documento HTML o XHTML o se escribe en un archivo separado que se enlaza al documento principal.

En conclusión, el desarrollo web está dominado por los frameworks client-side, como React, Angular o Vue, pero los de server-side ofrecen un desarrollo web E2E y, frameworks como Spring, ASP.NET Core, Django, Laravel, Phalcon o Ruby on Rails, están adquiriendo una tendencia de uso importante para back-end.





04. Movilidad

APLICACIONES NATIVAS



En las aplicaciones nativas **los desarrolladores no tienen control total del ciclo de vida de la aplicación ya que el sistema operativo puede llegar a interferir con la misma**, ya sea enviando notificaciones para que se bajen los niveles de uso de memoria o para pasar la aplicación a segundo plano. Por ello se debe estar preparado para cada uno de estos escenarios y responder acordemente a ellos, y así evitar perdida de información y asegurar el funcionamiento óptimo.

Tanto en iOS como en Android, para la implementación de los componentes, exceptuando a las clases que se construyen para cada aplicativo, es el propio SO el que ofrece sus componentes para que se utilicen en las aplicaciones.

ANDROID NATIVO

La **arquitectura MVVM** es el patrón de arquitectura más utilizado en Android y recomendado desde Google para la implementación de sus aplicaciones móviles. Permite separar la lógica de negocios de la interfaz de usuario, facilitando las pruebas, mantenimiento y la escalabilidad de los proyectos.

Los componentes de la arquitectura son **Model, View y ViewModel**, este último es el que se encarga de almacenar y administrar los datos relacionados con la IU de manera optimizada para que se conserven los datos después de cualquier cambio de configuración, como es la rotación de pantalla.

Los **Architecture Components** son una colección de bibliotecas que ayudan a diseñar apps más sólidas por medio de la utilización de librería proporcionadas por Android, lo que permite ahorrar esfuerzos de implantación y aporta la fiabilidad de ser elemento base de la arquitectura.

Están integradas en el core de Android, por lo que permiten interactuar con el ciclo de vida de forma natural y realizar acciones como respuesta a un cambio en el estado del ciclo de vida de otro componente, como puede ser una actividad o un fragmento. Esto permite crear un código mejor organizado, y a menudo más liviano, que resulta más fácil de mantener.

RECOMENDACIÓN

Aun que es recomendable utilizar MVVM en la mayoría de los proyectos, el **patrón MVP** sigue siendo un patrón perfectamente válido ya que permite utilizar el resto de los componentes de la arquitectura. Probablemente en MVP el uso de ViewModel no tiene sentido ya que es el sustituto natural del presentador, pero el resto de los componentes se pueden usar de una forma u otra y permite construir una aplicación con una baja complejidad.

Por lo tanto, aunque dependiendo del tipo de aplicación, se recomienda la utilización de la **arquitectura MVVM** que es la que mejor se adapta, en combinación de la utilización de **Architecture Component**. Este duo proporciona grandes ventajas:

La arquitectura se adapta a la perfección en el ecosistema Android.

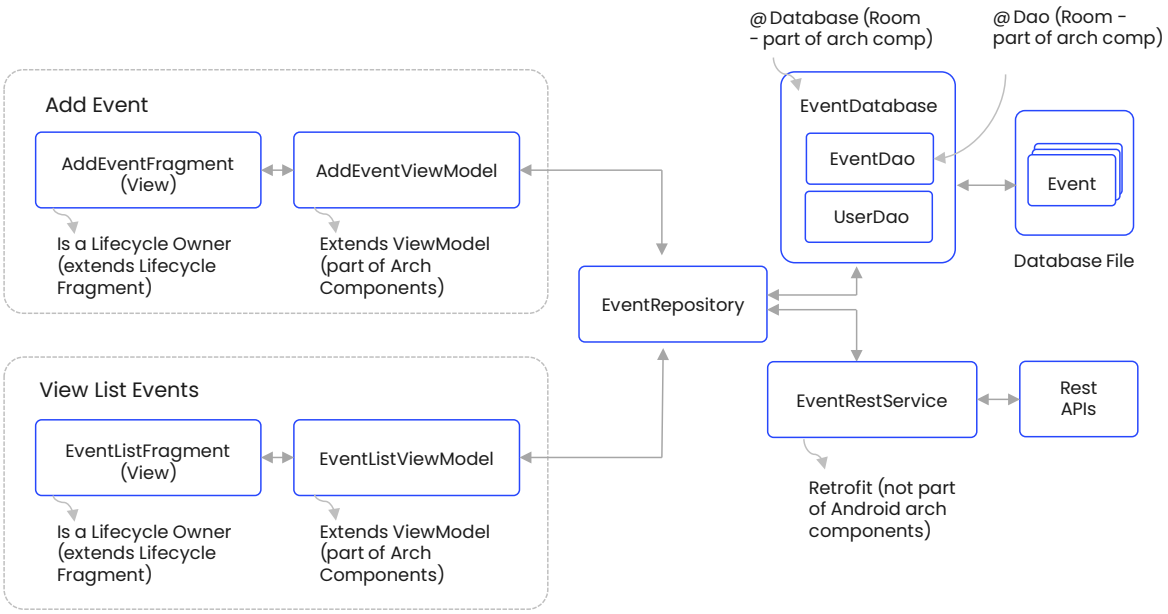
Proporciona una App más robusta.

Código más limpio y más legible.

Permite realizar test unitarios y de interfaz de usuario de una manera muy sencilla, puesto que el código está separado.

La aplicación que puede ser expandida y mantenida fácilmente, puesto que al utilizar MVVM se programa por capas.

MVVM y Architecture Component con Android:



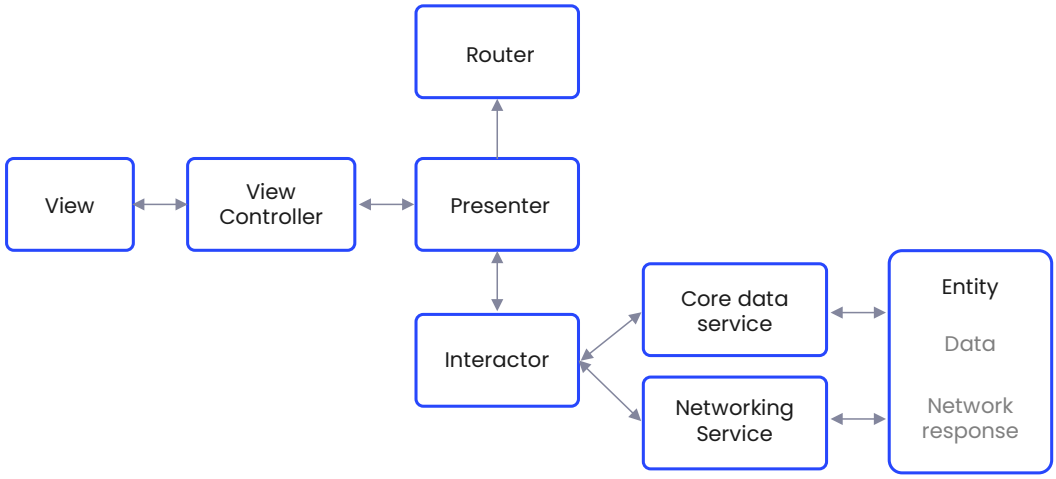
IOS NATIVO

Las aplicaciones iOS se componen principalmente de Storyboards, XIBs, Viewcontrollers y clases. Es recomendable utilizar el lenguaje **Swift** ya que es el más moderno y actualizado que Apple proporciona, y la arquitectura VIPER ya que esta provee la mejor separación de responsabilidades en un entorno iOS

VIPER es un patrón de arquitectura como MVP o MVVM pero este separa más el código siguiendo el principio de responsabilidad única. Se divide en 5 capas y cada una de sus letras corresponde a un componente de la arquitectura: View, Interactor, Presenter, Entity y Router.

Solo el presentador habla con la vista y solo el interactor habla con el modelo (entidad). El presentador y el interactor se coordinan entre sí. El presentador se preocupa por la visualización y la acción del usuario, y el interactor se ocupa de manipular los datos.

Patrón VIPER:



En cuanto a **librerías** para iOs, las más extendidas por la comunidad de desarrollo son:

ALAMOFIRE: Librería para las conexiónes con el servidor, facilita mucho la creación de Request.

KINGFISHER: Es una biblioteca ligera para descargar y almacenar en caché imágenes de la web. La descarga y el almacenamiento en caché se realizan de forma asincrónica.

SWIFTLINT: Herramienta de análisis de código estático.

RXSWIFT: Es un framework escrito en Swift para la programación radioactiva. Es una versión Swift de Rx y su objetivo es permitir una composición sencilla de operaciones asincrónicas y flujos de datos/eventos.

RECOMENDACIÓN

La arquitectura tradicional MVC y MVVM tiene un problema principal: los controladores gigantes. Dado que toda la lógica empresarial se escribe en el controlador, al final, el controlador es enorme y complejo, por los que se vuelve difícil de administrar para proyectos escalables. Es por ello por lo que se necesita VIPER.

Esta arquitectura es muy limpia, **aísla cada módulo de los demás**. Por lo tanto, cambiar o corregir errores es muy fácil, ya que solo tiene que actualizar un módulo específico. Esta modularidad ofrece un buen entorno

para las pruebas unitarias y mantiene el acoplamiento bajo. Además, dividir el trabajo entre los co-desarrolladores también es bastante simple lo que permite definir la propiedad por equipos de trabajo.

VIPER debe usarse cuando los requisitos de una aplicación están muy bien formados. Trabajar con requisitos en constante cambio puede crear confusión y códigos desordenados. Por lo tanto, no debe usarse en proyectos pequeños, ya que MVP o MVC serán suficientes. Además, VIPER solo debe usarse si todo el desarrollador del proyecto comprende completamente el patrón.



APLICACIONES HÍBRIDAS

Las aplicaciones móviles híbridas se construyen como una aplicación web, ya que sus fundamentos son HTML, CSS y JavaScript. Pueden compilarse para ser visualizadas como una aplicación web en Internet o integradas en un web view, un visor, que se comunica con el dispositivo móvil gracias a una capa intermedia en el que van embebidas. Y esta es su virtud principal, que **el grueso del código sirve para las distintas plataformas**.

React Native, ionic y Flutter son los principales frameworks para el desarrollo multiplataforma, cuyo fin es el desarrollo de aplicaciones móviles. Estos se encargan de reducir el proceso de desarrollo y diseño, que tradicionalmente puede resultar más tedioso.

React



React Native es un framework que implementa una jerarquía de componentes de UI para construir el código JavaScript que permite crear aplicaciones renderizadas de forma nativa para iOS y Android. Uno de sus puntos fuertes es que se creó en base a React y que dicha tecnología se ha ganado la confianza de empresas como Facebook, Skype, Instagram o Uber.

Las **principales fortalezas** de React Native son:



Reutilización de código.



Recarga en vivo y en caliente.



Arquitectura modular.

En general, se puede decir que la arquitectura de React Native ayuda a estructurar un proyecto para una aplicación móvil multiplataforma **manteniendo la lógica del negocio de una forma independiente y reutilizable**.

Para mejorar dicha arquitectura es recomendable el uso de la centralización de la información mediante el **patrón Redux**. Esta tecnología ayuda a que cualquier aplicación sea mucho más escalable a lo largo de su vida.

Redux es una biblioteca de JavaScript que permite controlar el estado de la aplicación. Lo realiza mediante un lugar centralizado llamado *store* donde se almacena y se modifica el estado mediante el uso de *actions* y *reducers*. Gracias a todo ello, se puede conseguir una única fuente de la verdad a la que los componentes se pueden conectar para obtener la información que necesiten.



Ionic es el **framework dominante** en todos los desarrollos a lo que híbrido se refiere. Respaldado en sus inicios por Angular y con varias versiones consolidadas a su espalda, sigue siendo una opción bastante lógica a la hora de plantear una solución técnica para cualquier Progressive Web App.

Ionic transformó completamente su orientación al pasar de ser un lenguaje basado principalmente en ser un framework web con un wrapeo nativo a **convertirse en una librería compatible con cualquier tecnología web** con el fin de poder incorporarse en cualquier proyecto.

Tras su desvinculación con Angular, la incorporación de StencilJS y la **nueva capa de comunicación que permite separarse de Cordova**, hace que Ionic, desde su versión 4, abra un gran campo en su metodología de trabajo.

Estos avances inducen a utilizar una nueva forma estructurada de exponer la funcionalidad nativa gracias a ***Capacitor***, que considera la carpeta de platforms como source asset. Es decir, se incorpora su contenido en el repositorio git para que se tenga más control sobre dicho código, esto da ventaja respecto al uso previo de Cordova.

Además de esto, ahora se pueden integrar las propias etiquetas de Ionic dentro de otros frameworks, por ejemplo React. Con el simple hecho de instalar "ionic/cli" y lanzar un proyecto de tipo react, se obtendrán las ventajas que bridan ambas tecnologías.



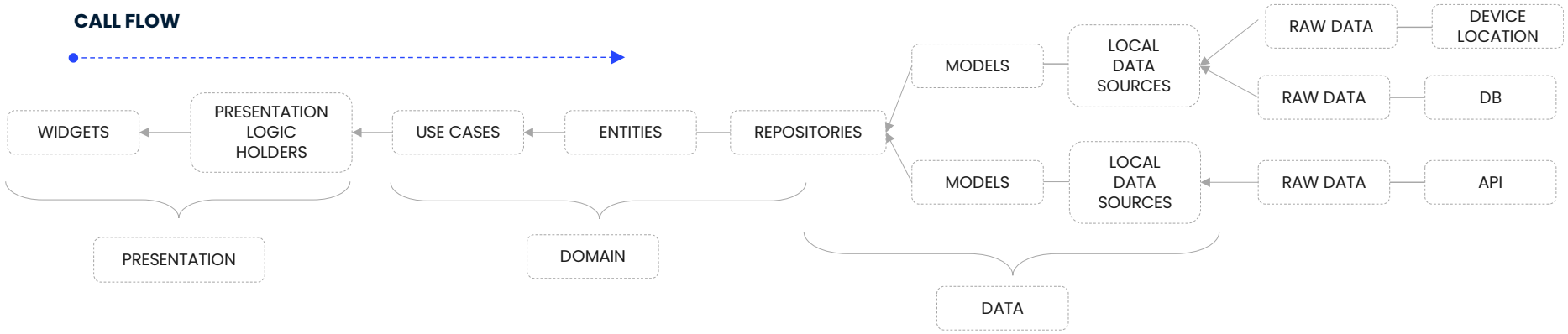
La **principal ventaja que muestra Flutter es su base sólida** gracias al equipo de desarrollo de Google. Proporciona un SDK que permite crear aplicaciones tanto para Android como para iOS siempre compiladas y con la generación de código 100% nativo.

Es recomendable utilizar el **patrón BLoC** (Business Logic Component) ya que ofrece multitud de ventajas a la hora de reutilizar código y facilita la administración del estado y el acceso a los datos desde un lugar central del proyecto. Comparado con otras arquitecturas, se puede decir que BLoC es el ViewModel en MVVM.

Se puede resumir como un componente intermediario que gestiona la comunicación entre flujo de eventos. Todo ello llevará a una mayor separación entre la lógica de negocio y la interfaz de usuario y una centralización para los cambios de estado.

Del mismo modo que trabaja Redux, BLoC se fundamenta en el flujo de datos unidireccional. La idea es que cada vista o componente tenga asociado su correspondiente BLoC y que éste actúe como una caja negra a la hora de su implementación.

Ejemplo de arquitectura de una app híbrida con Flutter:





Ventajas y desventajas de React Native, Ionic y Flutter:

	React	Ionic	Flutter
Ventajas	Tecnologías web (HTML, js, ts, css). Webcomponents. Genera código nativo. Fuerte comunidad. Creado por Facebook.	Tecnologías web (HTML, js, ts, css). Gran comunidad y soporte. Evolución continua.	Rendimiento. Creado por Google. Ecosistema propio. Librería de componentes basada en Material Design.
Desventajas	Utiliza plugins de la comunidad para obtener la funcionalidad de la plataforma. Tarda tiempo en actualizarse con las ultimas funcionalidades de los SO.	Dependencia de plugins de terceros. Alto tiempo en adoptar las últimas funcionalidades de las plataformas. No genera componentes nativos.	Lenguaje Dart. Al ser el más nuevo faltan algunas funcionalidades.

			
Framework	Framework JavaScript.	Framework de desarrollo de aplicaciones móviles HTML5.	Framework de UI móvil de Google.
Lenguaje de programación	JavaScript+, Swift, Objective-C o Java.	HTML5, CSS y JavaScript+ o Typescript.	Dart.
Rendimiento	Casi nativo.	Moderado.	Muy bueno.
GUI	Utiliza controladores de UI nativos.	HTML y CSS.	Utiliza wiggets propios y ofrece una muy buena UI.
Apoyo del mercado y de la comunidad	Una gran comunidad y soporte online de expertos en Java Script que pueden ofrecer soluciones efectivas a cualquier problema.	Proporciona apoyo y guía para construir la app móvil.	Es relativamente nuevo, pero hay tutoriales y guías disponibles en Internet.
Reutilización del código	El 90% del código es reutilizable. Permite al usuario reutilizar el código, pero sólo proporciona los componentes básicos para empezar a reestructurar.	El 98% del código es reutilizable. Viene con una amplia biblioteca de componentes adaptables que ayuda a los usuarios a reutilizar su código para el restyling y las plataformas subyacentes.	El 50-90% del código es reutilizable. Permite volver a utilizar lo escrito. Es ideal si se desea volver a diseñar la plataforma de aplicación móvil específica.
Aplicaciones populares	Facebook, Instagram, Airbnb, UberEats.	Justwatch, Pacífica, Nationwide, Marketwatch.	Alibaba, Google Ads, Tencent, HamilTon.
Precio	Open-source.	Open-source y de pago.	Open-source.
Recarga en caliente	Si	No	Si
Tamaño de la aplicación	Media	Grande	Grande
Curva de aprendizaje	Difícil	Fácil	Fácil



05. **Bases de Datos**

Las bases de datos deben evolucionar y adaptarse a medida que crece el Big Data. Las bases de datos SQL tradicionales como Oracle o MySQL no pueden distribuir datos y consultas de forma automática en varias instancias. Las bases de datos NoSQL han aparecido en el mercado para solucionar las limitaciones de escalado que presentan las bases SQL. Aun así, SQL se ha llevado los primeros puestos de las bases de datos más populares en 2020, principalmente porque estas ofrecen garantías ACID.

La clave de éxito de una base de datos seria deshacerse de las bases SQL manteniendo la garantía ACID, es decir, mantener dicha garantía junto con la mejora de rendimiento y escalabilidad que presentan las bases de datos NoSQL.

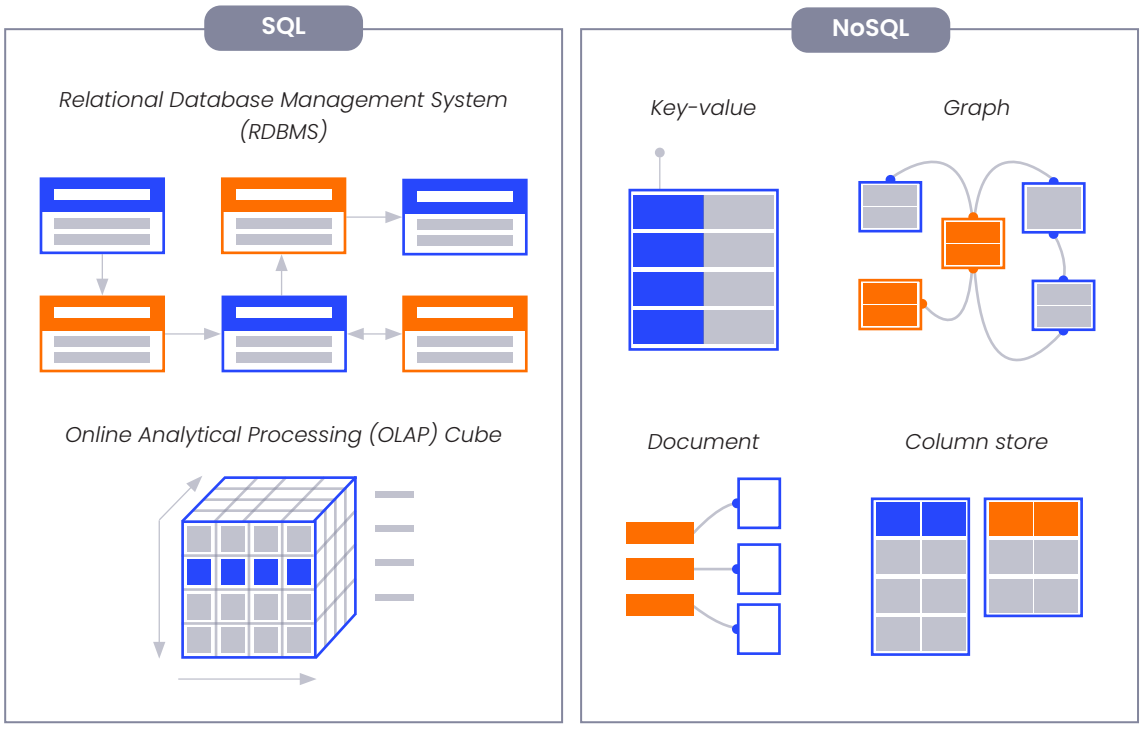


DIFERENCIAS SQL Y NOSQL

La principal diferencia está en el **tipo de propiedades que caracterizan a una base de datos**: mientras que las SQL presentan garantía ACID, que son atomicidad, consistencia, aislamiento y durabilidad; las NoSQL siguen el teorema Brewers CAP, basada en la consistencia, disponibilidad y tolerancia a la participación.

Respecto al **tipo de almacenamiento**, las NoSQL permiten un almacenamiento jerárquico, mientras que las SQL no dan esta posibilidad. El almacenamiento jerárquico es posible por el método Key-Value que siguen las bases NoSQL, ideal para grandes volúmenes de datos.

Comparación de esquemas SQL y NoSQL:



Otra diferencia fundamental entre ambas es el **tipo de escalado** que ofrecen. Las bases de datos NoSQL presentan un escalado horizontal pudiendo agregar servidores adicionales a su infraestructura, y las bases de datos SQL son escalables verticalmente, es decir, que se puede administrar el crecimiento de carga aumentando el procesador en un único servidor. En definitiva, las bases NoSQL también permiten la realización de transacciones complejas, pero no es comparable a como lo hacen las bases de datos SQL. Las consultas complejas son idóneas para las bases de datos SQL, ya que se adaptan muy bien a grandes volúmenes de consultas. Por el contrario, las NoSQL presentan una mayor dificultad en este aspecto.

SERVICIOS DE BASES DE DATOS



Por ahora, lo más parecido a este tipo de **bases de datos que aúnen lo mejor de ambos mundos son Amazon's Aurora y Google's Spanner**, pero cada una presenta sus inconvenientes. Amazon's Aurora no es capaz de ofrecer un escalado horizontal, mientras que Google's Spanner sí, pero carece de características fundamentales y necesarias de las bases de datos SQL. En cuando a **bases de datos No SQL, hay que mencionar Amazon DynamoDB y Google BigTable**.

Relational Database Service (RSD) es el encargado de administrar **Amazon Aurora**, que se define como un motor de búsqueda compatible con MySQL y PostgreSQL. Es una base de datos creada para la nube, se caracteriza por su simplicidad y un precio económico. Muestra un elevado rendimiento similar al de las bases de datos empresariales tradicionales, mejorando así las funciones de MySQL y PostgreSQL.

Aurora incluye un subsistema de almacenamiento de rápida distribución. Su arquitectura funciona mediante la formación de clusters y continua replicación de la información, así como el crecimiento del almacenamiento subyacente.

Cloud Spanner ofrece coherencia transaccional y escalado en un solo servicio totalmente administrado. Por tanto, es una base de datos distribuida que ofrece disponibilidad y escalado horizontal. Estas ventajas, acelerarán la capacidad de llevar soluciones a las cadenas de suministro omnicanal basadas en la nube, por ejemplo.

Además, el escalado horizontal se puede realizar sin particionamiento complejo ni agrupación en clusters. Spanner dispone de cifrado de capa de datos, administración de identidades y accesos y el registro de auditoría, para evitar errores.

Cloud Spanner, también es bastante económico, ya que se paga con precios determinados por horas de nodo de proceso, consumo de almacenamiento real y acceso a la red externa. Además, ofrece bibliotecas de cliente para los lenguajes más populares, incluyendo Java, Go, Python y Node.js.

SQL & Compatibilidad de las transacciones	PostgreSQL & MySQL (compatibilidad total).	SQL propio (sin claves externas).
Conmutación por error/ reparación nativa	✓	✓
Escalabilidad de escritura horizontal	✓ Multi-master	✓ Auto-sharded
Distribución geográfica de los datos	✓ Sólo una región puede tener escrituras.	✓ Sincronización global del reloj para mayor coherencia.
Alto rendimiento	✗ Las réplicas de lectura no pueden procesar las escrituras.	✗ no está optimizado para la ingesta de grandes volúmenes.
Cloud Neutral con Kubernetes Nativo	✗ Propiedad de AWS.	✗ Propiedad de Google Cloud.
Open Source	✗	✗





DynamoDB es un servicio de base de datos NoSQL completamente administrado en la nube. Permite transferir las cargas administrativas de las bases de datos distribuidas (provisionamiento, instalación, configuración, etc.) y ofrece gran escalabilidad y rendimiento. Además, es compatible con almacenes clave-valor y bases de datos de documentos.

Ofrece cifrado automático, backup bajo demanda y copias de seguridad bajo demanda que permiten la recuperación de datos. Además, es posible la monitorización de recursos y métricas de desempeño a través de la consola de administración de AWS.

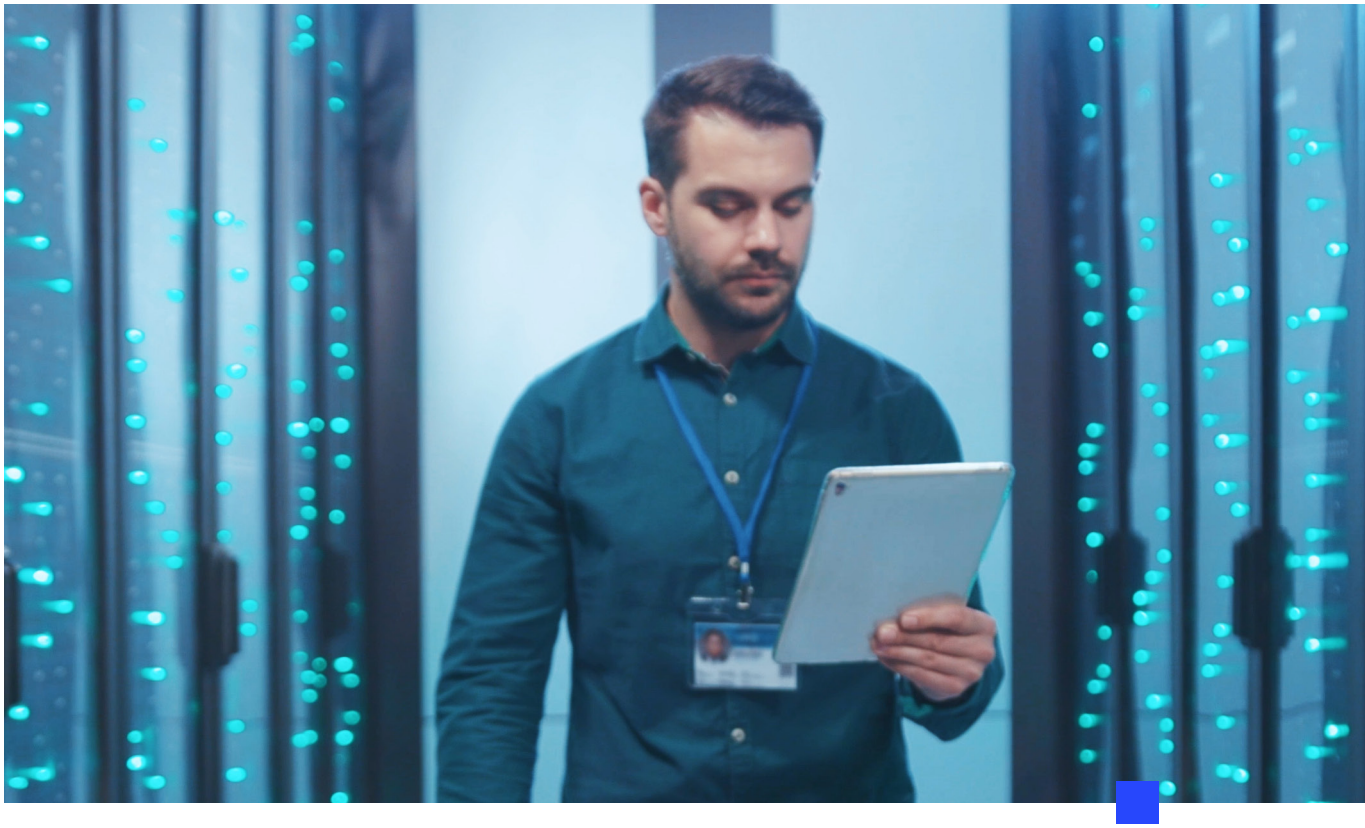
Es el producto estrella de AWS (si no contamos S3) y es líder indiscutible en cuanto a la relación rendimiento/coste en BBDD noSQL gestionadas. Su flexibilidad y desempeño lo hacen ideal para RR.SS., juegos, publicidad o IoT, entre otros.

Google Bigtable es el servicio de bases de datos NoSQL de Google Cloud, rápido, escalable, eficiente y totalmente administrado. Además, se integra fácilmente con herramientas de macrodatos.

Ofrece baja latencia, alta capacidad de procesamiento, cambio de tamaño de clústers sin inactividad, optimización de la carga de trabajo (de operaciones o analítica), con replicación automatizada y flexible.

Fue creado para cargas de trabajo muy elevadas por lo que es muy útil para IoT, sector financiero, publicidad y personalización, entre otros.

		
Arquitectura	Descentralizado	Centralizado
Modelo de datos	Key-value	Mapa multidimensional ordenado
API	get, put	get, put, scan, delete
Seguridad	No	Control de acceso a nivel de familia de columnas
Partición	Hashing consistente	Basado en el rango de claves
Replicación	Nodos sucesores en el anillo	Chunkservers en GFS
Almacenamiento de datos	Objetos de valor binario	Colum stores (SSTables)
Afiliación y detección de fallos	Protocolo Gossip	Handshakes iniciados por master
Gestión de datos	Estructurado y no estructurado	Estructurado
Fuente de datos	Binarios, cadenas, números	Cadenas, gráficos
Orientación del sistema	Simétrico	Sparse
Distribución de la carga	Load sharing (heterogéneo)	Load balancing
Escalabilidad	Incremental	Alta
Consistencia	Eventual	Alta
Control de datos	Estrecho	Dinámico
Lenguaje	Java, .Net, Javascript, Perl, C#	JACOB
Aislamiento del sistema	Si	No
Tolerancia a los fallos	No	Si
Plataforma	EC2	Linux, MAC, Windows



CONCLUSIÓN

Las bases de datos SQL y NoSQL son capaces de llevar a cabo el almacenamiento y la recuperación de datos de forma optimizada. Las bases de datos distribuidas NoSQL se han vuelto habituales en el día a día de diferentes empresas. Sin embargo, fuerzan opciones de diseño difíciles dejando de lado la garantía ACID requerida en la actualidad.

Ahora, existe una nueva clase de bases de datos, las bases de datos SQL distribuidas, que persiguen aunar las ventajas de una base de datos NoSQL con la garantía ACID de las SQL. El mercado dispone ahora mismo de bases de datos como Amazon's Aurora o Google's Spanner capaces de aproximarse a estos requisitos deseados dentro de la evolución del Big Data. Y en cuanto a NoSQL destacan Amazon DynamoDB y Google BigTable.

Es fundamental que cada empresa seleccione el producto disponible en el mercado que más se adapte a sus necesidades, ya que no existe una base de datos perfecta en todos los ámbitos y dependerá de la aplicación requerida.

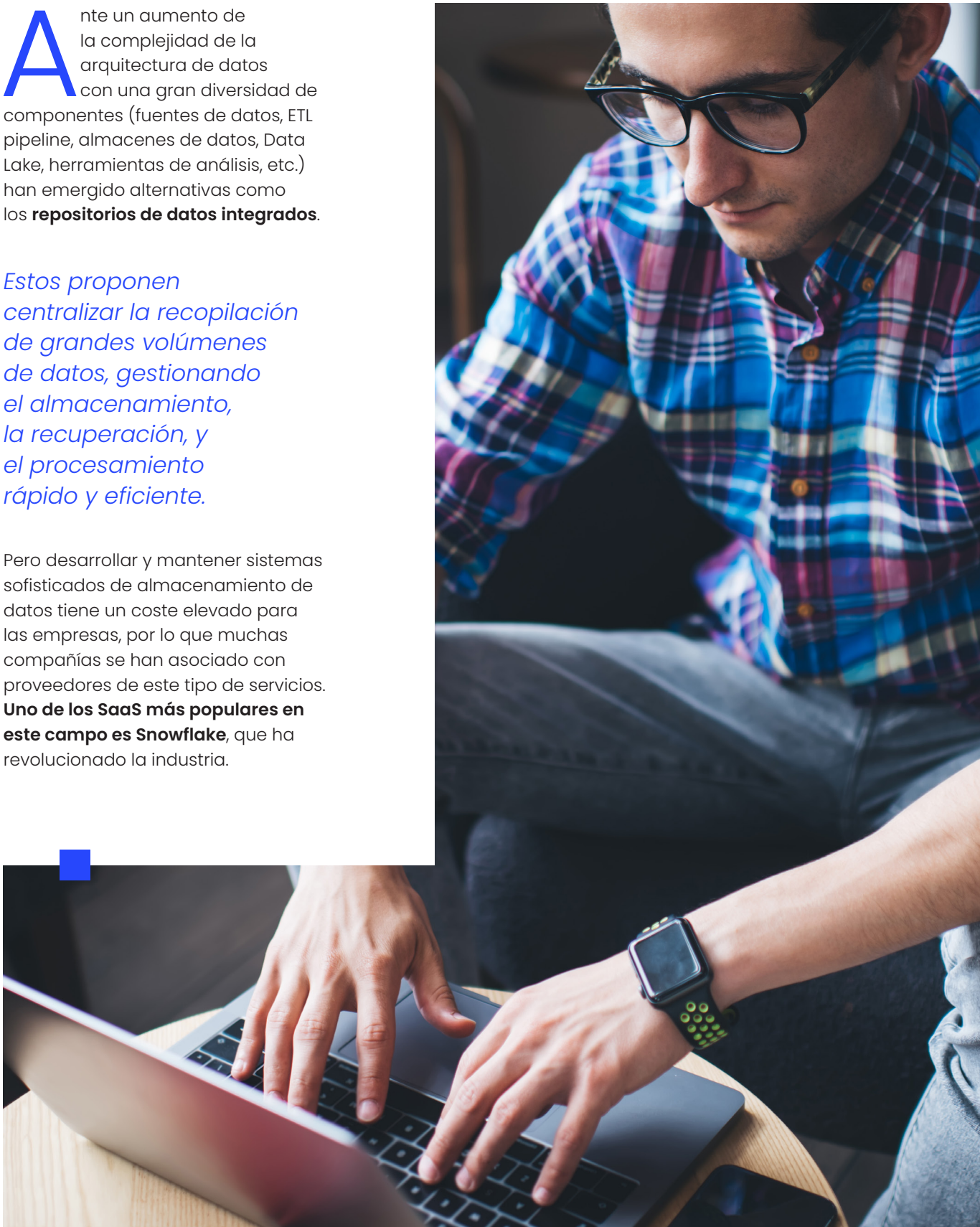


06. Data Platform

Ante un aumento de la complejidad de la arquitectura de datos con una gran diversidad de componentes (fuentes de datos, ETL pipeline, almacenes de datos, Data Lake, herramientas de análisis, etc.) han emergido alternativas como los **repositorios de datos integrados**.

Estos proponen centralizar la recopilación de grandes volúmenes de datos, gestionando el almacenamiento, la recuperación, y el procesamiento rápido y eficiente.

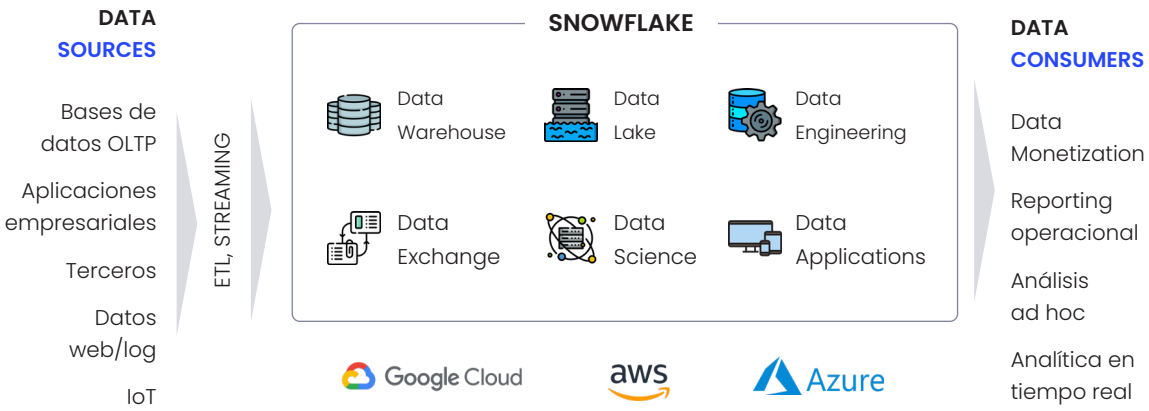
Pero desarrollar y mantener sistemas sofisticados de almacenamiento de datos tiene un coste elevado para las empresas, por lo que muchas compañías se han asociado con proveedores de este tipo de servicios. **Uno de los SaaS más populares en este campo es Snowflake**, que ha revolucionado la industria.



Se define a Snowflake como un **almacén de datos basado en entornos cloud** (principalmente Microsoft Azure o AWS), evitando de este modo el uso de servidores propios, y ahorrando costes para las compañías a la hora de implementar almacenes de datos.

Snowflake destaca porque, a diferencia de las arquitecturas clásticas, existe solo una "Source of Truth" de datos. Con ello, es capaz de proporcionar una fuente de datos centralizada a la par que construye cargas de trabajo dentro de ella.

Este SaaS está diseñado para gestionar cargas de trabajo de manera simultánea, desde Data Lake al uso compartido de datos. Dispone de una arquitectura y capacidad de intercambio de datos única, ejecutándose en infraestructuras de nube pública. Además, Snowflake es una plataforma que **no necesita demasiado mantenimiento y se comercializa bajo una fórmula de pago por uso**.



De este modo, Snowflake se caracteriza por ser una **plataforma unificada para gestionar y administrar cargas de trabajo**, y es compatible con una amplia gama de herramientas de integración y procesamiento de datos, que le confieren la capacidad de implementar almacenes virtuales en milisegundos. Estos almacenes trabajan únicamente en la nube, por lo que poseen una potencia de cómputo prácticamente infinita y un almacenamiento de datos ilimitado.

De manera adicional, la plataforma posee la capacidad de escalar verticalmente el almacén virtual y procesar terabytes de datos con facilidad. Así pues, es compatible con soluciones de aprendizaje automático y, al ser una plataforma que no requiere prácticamente mantenimiento, hace que sea **una solución muy aconsejada para el ámbito de la IA**.



ARQUITECTURA DE SNOWFLAKE

La arquitectura de Snowflake es una combinación de arquitecturas que aprovecha las ventajas de un escalado horizontal, dando como resultado un rendimiento mejorado. Utiliza un repositorio de datos central para los datos persistentes a los que se puede acceder desde todos los nodos de la plataforma y, al mismo tiempo, procesa consultas mediante clústeres.

Está dividida en 3 capas, cada una con sus dimensiones escalables:

1. ALMACENAMIENTO DE BASES DE DATOS

Snowflake reorganiza los datos que se cargan en su formato interno. A continuación, almacena estos datos optimizados en la nube. Es decir, controla la organización, el tamaño del archivo y la estructura de los datos. Los objetos de datos almacenados por Snowflake solo son accesibles a través de operaciones de consulta.

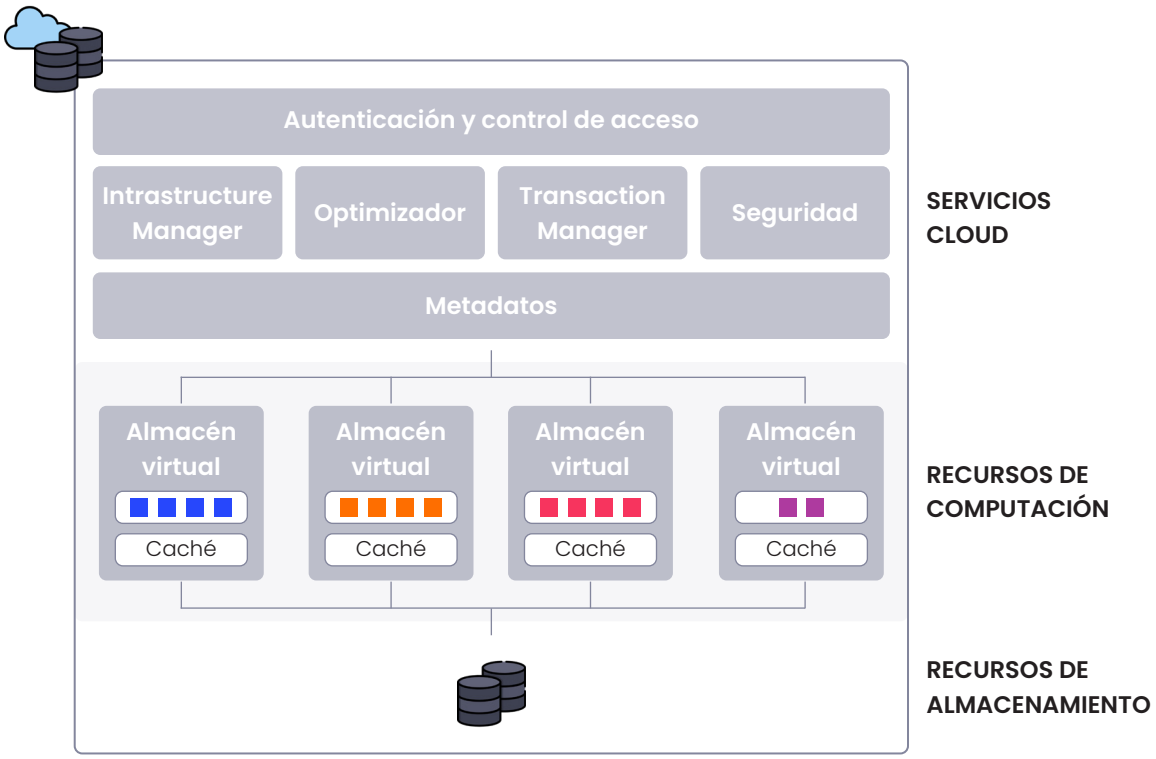
2. CAPA DE CÁLCULO

Snowflake procesa las consultas mediante almacenes virtuales, de tal forma que se permite un escalado automatizado sin la obligación de redistribuir los datos de la capa de almacenamiento, puesto que la capa de cálculo opera sobre almacenes automatizados.

3. SERVICIOS EN LA NUBE

La capa de servicios en la nube tiene como fin coordinar actividades de forma que se procesen las solicitudes de usuario desde el inicio de sesión hasta el envío de consultas. Los servicios administrados en esta capa incluyen: autentificación, gestión de infraestructuras, gestión de metadatos, análisis, optimización de consultas y control de acceso.

Arquitectura de Datos Compartidos del Multi-Cluster Snowflake



Sistema único e integrado



VENTAJAS DE SNOWFLAKE

Snowflake permite soluciones de análisis y almacenamiento de datos más rápidas, fáciles de usar y más flexibles que las ofertas tradicionales.

Su nube de datos está impulsada por una plataforma que **no depende de ningún software de base de datos existente**, y aun así ofrece el usuario todas las funcionalidades de una base de datos empresarial.

Entre los beneficios más comunes de Snowflake está la **mejora del rendimiento**, ya que trabajar en la nube aporta elasticidad, pudiendo ampliar el tamaño del almacén, cargar datos o ejecutar una gran cantidad de consultas de un modo más rápido. Además, emplea el lenguaje de consulta SQL que optimiza el funcionamiento de los equipos.

Paralelamente, se encarga de todos los aspectos de la instalación, así como de las actualizaciones del software. Además, **se puede usar con cualquier proveedor de Cloud principal**, porque es un programa independiente.

Entre otros beneficios, los usuarios de Snowflake pueden **crear un número ilimitado de repositorios virtuales**, y cambiar rápida y fácilmente el número de nodos en cada grupo para obtener un rendimiento óptimo.

Este almacén de datos aporta la posibilidad de **combinar datos estructurados y semiestructurados** para procesarse en la nube sin tener que traducirse a esquemas de relación fijados. De este modo, optimiza la forma en que se procesan y consultan automáticamente los datos. Por último, facilita el intercambio de datos entre los usuarios y con el consumidor.

PRINCIPALES COMPETIDORES

Grandes empresas tecnológicas como Microsoft, Amazon, Google (Alphabet), SAP, Oracle e IBM tienen plataformas de almacenamiento de datos basadas en la nube que compiten directamente con Snowflake, y no es de extrañar, teniendo en cuenta el papel que desempeñan los datos en la inteligencia empresarial. Así, los principales competidores de Snowflake son **Microsoft Azure Synapse Analytics, Google Cloud BigQuery, Amazon Redshift (parte de AWS) o IBM Db2 Database**.



No obstante, Snowflake ha revolucionado el mercado, dadas sus ventajas sobre el resto de los competidores y tendrá un papel protagonista en la democratización del acceso a almacenes de datos.



07. Data-Intensive Computation

Una de las principales problemáticas actuales es que se generan muchos más datos de los que se es capaz de analizar. Para solucionar esto surgió el Data Intensive Computing, que **utiliza el paralelismo de datos para el procesamiento óptimo de grandes conjuntos de datos**.

ALGUNAS DE SUS PRINCIPALES CARACTERÍSTICAS SON:



Los algoritmos se ejecutan en donde se encuentran los datos, reduciendo su movimiento y la sobrecarga del sistema, aumentando así el rendimiento.



Utiliza un enfoque independiente de la máquina en el que el sistema de tiempo de ejecución es el que se encarga de controlar la ejecución, programación, comunicaciones, equilibrio de carga y el flujo de datos.

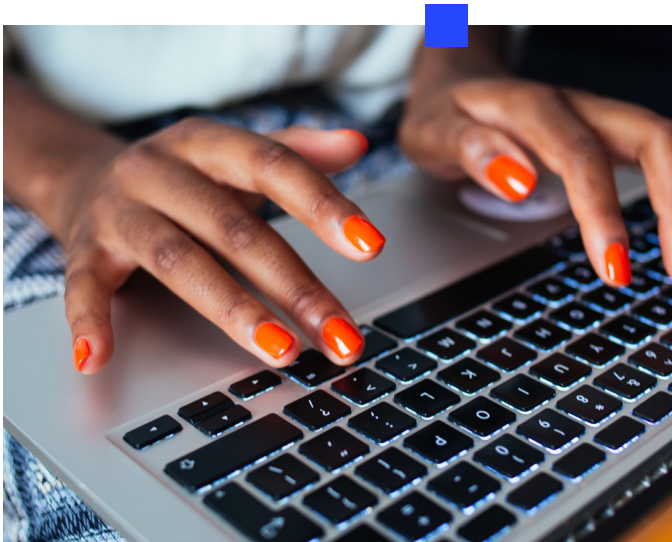


Aporta mayor disponibilidad y fiabilidad que los sistemas tradicionales a gran escala, ya que estos son más susceptibles a fallos.



Ofrece gran escalabilidad, permitiendo adaptarse tanto a una gran cantidad de datos como a los requisitos del tiempo de rendimiento.

Anteriormente, se utilizaba el framework Hadoop MapReduce para el procesamiento de grandes cantidades de datos, pero poco a poco ha quedado obsoleto, y se creó Apache Spark para solucionar sus limitaciones.



Apache Spark es un cluster de procesamiento distribuido open source que utiliza el almacenamiento en caché de la memoria y una ejecución optimizada de consultas para analizar cualquier volumen de datos. Además, proporciona una API de desarrollo en Java, Scala, Python y R, y admite la reutilización de código en varias cargas de trabajo.

BENEFICIOS DE APACHE SPARK

Permite escoger **diferentes lenguajes** para crear sus aplicaciones, ya sea Java, Scala, R o Python.

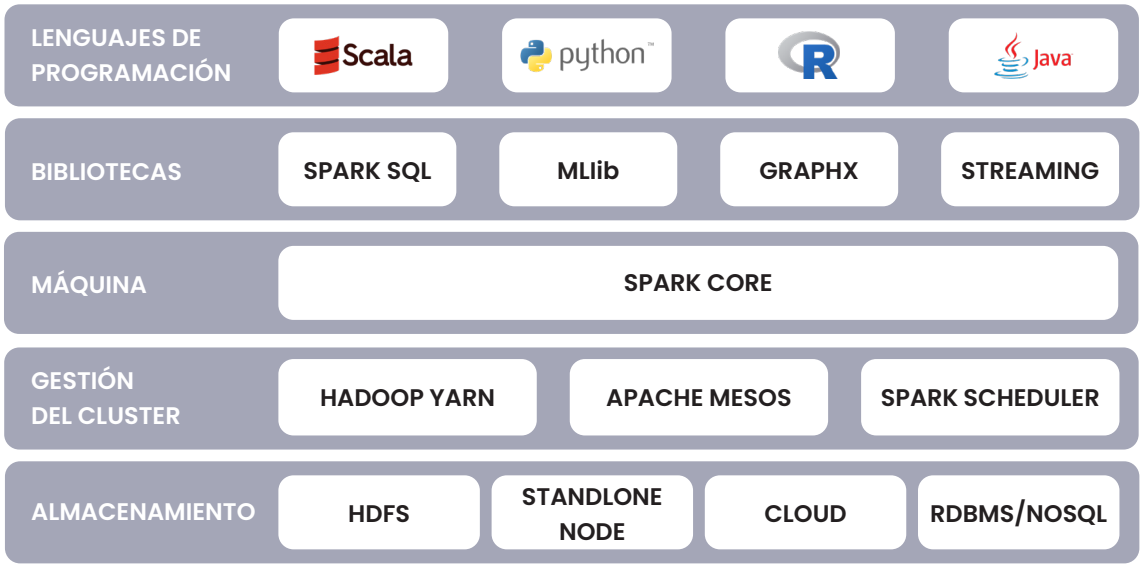
Puede ejecutar **consultas analíticas rápidas** con datos de cualquier tamaño mediante el almacenamiento en caché.

Es capaz de ejecutar **varias cargas de trabajo** de manera simultánea.

Incluye **bibliotecas** que aumentan la productividad con soporte para consultas SQL (Spark SQL), aplicaciones de streaming de datos (Spark Streaming), trabajar con machine learning de forma escalable (MLlib) y procesamiento de gráficos para crear flujos de trabajo complejos (GraphX).

Estas APIs ocultan la complejidad del procesamiento distribuido detrás de operadores simples de alto nivel. Por tanto, se **reduce la cantidad de código** necesario.

Apache Spark ecosystem



ARQUITECTURA Y EJECUCIÓN DE TAREAS

ENTIDADES PRINCIPALES:



Driver: desglosa el código en varias tareas que pueden ser distribuidas a través de los nodos.



Executors: ejecutan las tareas que el driver les asigna en los nodos.

CONCEPTOS CLAVE:



SparkContext: punto de entrada principal de Spark, donde se crean las variables que gestiona el framework.

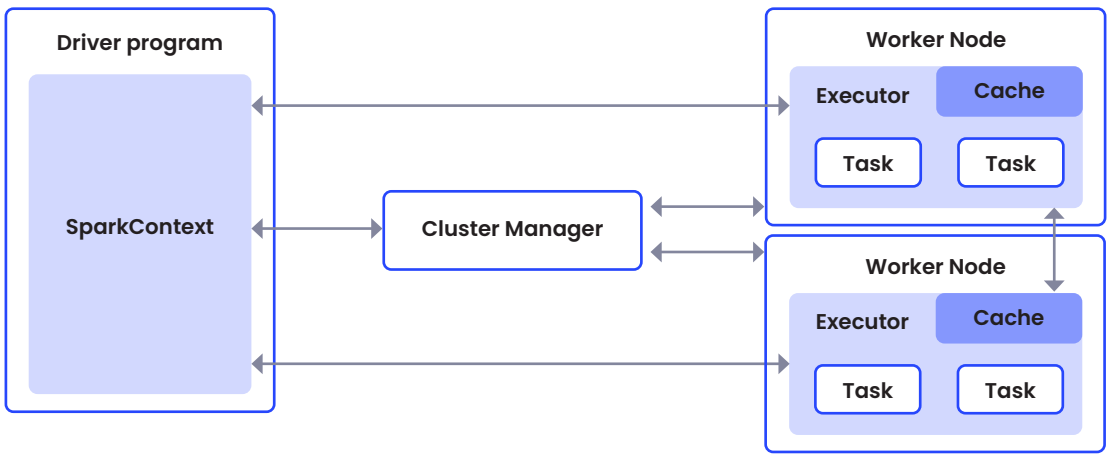


RDD: colección inmutable y particionada de elementos sobre los que se puede operar en paralelo.

Después de que una aplicación externa se inicie e inste una SparkContext comienza el **flujo de ejecución**:

1. El driver pide los recursos al administrador del cluster, que hace de mediador.
2. El Cluster Manager asigna los recursos entre las aplicaciones.
3. Spark adquiere executors en los nodos del clúster.
4. Se envía el código de la aplicación a los ejecutores.
5. SparkContext envía tareas a los executors.
6. Los executors ejecutan las tareas y almacenan el resultado.

En caso de error del Executor, las tareas se asignarán a otros Executors diferentes. Además, cada vez que un nodo se bloquea o se vuelve más lento, Spark inicia una copia preventiva de la tarea en otro Executor o nodo del clúster.



APACHE SPARK VS HADOOP MAPREDUCE

MapReduce quedó obsoleto debido a su velocidad insuficiente para la enorme carga de datos que soportan las aplicaciones hoy en día. En este contexto se presenta el desafío de reducir el número de pasos necesarios para ejecutar un proceso secuencial.

Spark se creó precisamente para abordar las limitaciones de MapReduce. Con Spark, solo se necesita un paso, donde los datos se leen en la memoria y las operaciones realizadas y los resultados se vuelven a escribir, lo que resulta en una ejecución mucho más rápida.



También, **Spark hace uso de DataFrames para la reutilización de datos mediante una memoria caché**, y conseguir así una abstracción sobre el conjunto de datos distribuidos. Con ello, se aceleran significativamente los algoritmos de machine learning, y se logra una mayor velocidad de procesamiento que con MapReduce.

Esta herramienta **se ha convertido en el proveedor neutral de la plataforma de Distributed Batch Jobs y ha superado al Sistema de Archivo Distribuido Hadoop (HDFS) en rendimiento y costes**. De hecho, se predice que Spark gobernará sobre el framework de procesamiento por lotes en 2021. No obstante, y con independencia de sus diferencias, Spark y Hadoop MapReduce pueden complementarse y trabajar conjuntamente.

Además, Spark también tiene sus limitaciones: es un sistema basado en JVM muy intensivo en recursos. Sin embargo, no parece existir un marco más optimizado en el horizonte cercano.

En resumen, Apache Spark ha desplazado a Hadoop en la gestión de grandes volúmenes de datos, caracterizándose por una mejora de rendimiento y una gran capacidad de trasmisión de datos, siendo el nuevo producto estrella dentro del ecosistema Apache y erigiéndose como nuevo estándar de la computación intensiva de datos.

		
Velocidad	100 veces más que MapReduce.	Más rápido que el sistema tradicional.
Facilidad de uso	Compacto y más sencillo que Hadoop.	Proceso complejo y largo.
Almacenamiento de datos	Almacenamiento de datos en memoria.	Almacena los datos en el disco.
Escrito en	Scala	Java
Procesamiento de datos	Batch / tiempo real / interativo / gráfico.	Batch processing.
Tolerancia a los fallos	RDD proporciona la tolerancia a fallos.	Hadhoop también es tolerante a fallos.
Machine Learning	MLlib sobre Spark proporciona algoritmos ML a gran velocidad.	Mahou sobre Haddoop proporciona algoritmos de ML.
Escalabilidad	Cluster de 8000 nodos operativos.	Hadoop ha sido probado en 15000 nodos.
Latencia	Proporciona computación de baja latencia.	Marco de computación de alta latencia.
Streaming	Procesa flujos de datos en tiempo real.	Procesa datos sólo en modo batch.
Almacenamiento en caché	Almacena los datos en memoria y mejora el rendimiento del sistema.	No admite el almacenamiento en caché de los datos.
Coste	Mayor que Hadoop.	Bajo coste.

DATABRICKS

Por ultimo cabe destacar la relevancia que ha tomado Databricks que con su última ronda de financiación ha alcanzado los 28 mil millones de dólares, encontrándose entre sus inversores AWS, Capital G (Google), Microsoft y Salesforce.

Databricks es una **plataforma de analítica de datos e Inteligencia Artificial basada en Apache Spark** y fundada por los creadores de Apache Saprk, Delta Lake y MLflow. Sus principales características son la simplicidad y que es open source y colaborativa.

Lo que añade esta plataforma sobre Spark es la fiabilidad y el alto rendimiento de los data wharehouse, y la apertura y flexibilidad que aportan los data lakes. Todo ello permite gestionar y almacenar datos estructurados, semiestructurados y no estructurados.

Además, ofrece **Notebooks** que permiten colaborar a Data Scientists, Data Engenieers y Data Analysts, para que trabajen de manera conjunta en tiempo real, comenten y compartan insights.

También permite utilizar herramientas de BI, consultando y visualizando los datos directamente en Databricks, compartir información con Dashboards en tiempo real, así como gestionar todo el flujo de ML en un entorno abierto y colaborativo con soporte para los principales frameworks.

LENGUAJES DE PROGRAMACIÓN

python™ Scala R Java SQL

LIBRERÍAS Y FRAMEWORKS

PyTorch TensorFlow Keras scikit learn GraphX

SISTEMAS DE CONTROL DE VERSIONES

GitHub ATLASSIAN Bitbucket

Cabe destacar que **tanto Microsoft Azure como Amazon Web Services (AWS) ya llevan tiempo ofreciendo Databricks** entre sus servicios cloud.





08. Streaming en Tiempo Real

En el punto en el que se encuentra la digitalización de los negocios, el streaming o análisis en tiempo real se vuelve fundamental para analizar rendimientos, detectar fraudes, optimizar búsquedas, etc. Cada día se necesitan interacciones más rápidas y eficientes, y precisamente el streaming permite comunicaciones agiles y detección de errores **para remediar con eficacia problemas que puedan surgir dentro de una compañía, o para proporcionar a los clientes la mejor experiencia de usuario**. Con ello, se está evolucionando de la tradicional arquitectura Lambda, simplificándola.

Así, el streaming consiste en obtener latencias de tiempos de procesamiento inferiores a un milisegundo, ayudando de este modo a tomar decisiones más rápidas. En resumen, proporciona agilidad en la filtración de datos dentro de las aplicaciones.

CARACTERÍSTICAS Y CONCEPTOS CLAVE

Existen algunas características y conceptos relacionados con el procesamiento en stream que se deben conocer para entender las fortalezas y limitaciones de cualquier framework:



GARANTÍAS DE ENTREGA

Existen diferentes niveles de garantía de que se procese un registro entrante concreto en un motor de transmisión:

- **At-most-once**: el registro entrante se procesará una vez o ninguna.
- **At-least-once**: el registro entrante se procesará al menos una vez incluso en caso de fallos. No habrá registros entrantes sin procesar.
- **Exactly-once**: el registro entrante se procesará una sola vez incluso en caso de fallos. No habrá registros entrantes sin procesar, ni tampoco registros duplicados. Aunque esto es lo deseable es difícil de lograr en sistemas distribuidos y se relaciona con el rendimiento.



TOLERANCIA A FALLOS

En caso de fallos, ya sean en los nodos, en la red, etc., el framework debería poder recuperarse y comenzar a procesar nuevamente funcionando de forma contínua desde donde se interrumpió el sistema. Esto se logra mediante la prevención generando redundancia, la detección mediante la monitorización y la recuperación devolviendo el sistema a condiciones operativas óptimas tras un fallo.



GESTIÓN DE ESTADO

En caso de requisitos de procesamiento de estado en los que necesitamos mantener algún estado (por ejemplo, recuentos de cada palabra distinta que aparece en los registros), el framework debería poder proporcionar algún mecanismo para preservar y actualizar la información de estado.



RENDIMIENTO

Esto incluye latencia (tiempo que se tarda en procesar un registro), rendimiento (registros procesados / segundo) y escalabilidad. La latencia debe ser lo mínima posible, mientras que el rendimiento debe ser lo mayor posible. Es difícil conseguir ambos al mismo tiempo.



FUNCIONES AVANZADAS

Event Time Processing, Watermarks y Windowing son funciones necesarias si los requisitos de procesamiento de secuencias son complejos. Por ejemplo, procesar registros según el momento en que se generaron en la fuente (procesamiento de tiempo de eventos).



MADUREZ

La madurez del framework es importante desde el punto de vista de la adopción, cuanto más probado esté, sobre todo por grandes empresas, mejor, ya que es más probable que obtenga un buen apoyo de la comunidad.



TIPOS DE PROCESAMIENTO DE STREAMS

Por otro lado, en cuanto al procesamiento, **existen dos enfoques para implementar un marco de transmisión**:

STREAMING NATIVO

También llamado procesamiento en tiempo real, consiste en que los datos se procesan en el momento en el que entra el registro, antes de que se almacenen. Este procesamiento o consultas se realiza en el registro de datos más reciente o en un periodo de tiempo continuo que, en función del marco, se llama operadores, tares o pernos.

EJEMPLOS

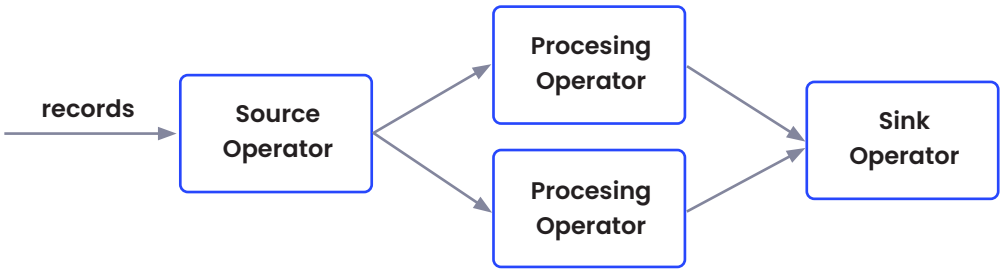
STORM

FLINK

KAFKA STREAMS

SAMZA

Registros procesados al momento:



MICRO-BATCHING

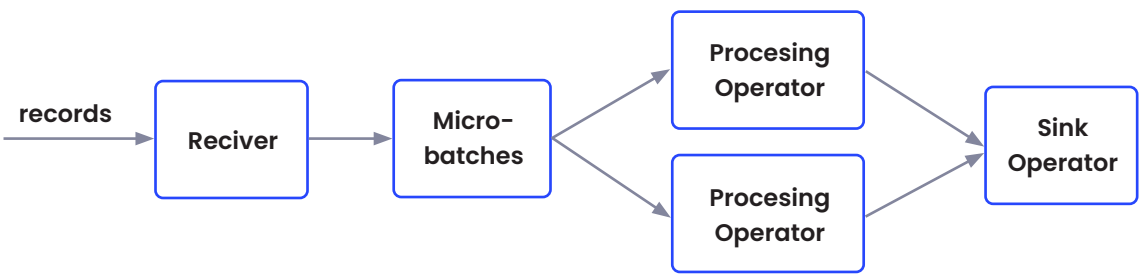
También conocido como Fast Batching, consiste en que los registros entrantes son agrupados cada pocos segundos y posteriormente se procesan en un solo micro lote con un retraso de unos segundos.

EJEMPLOS

SPARK STREAMING

STORM-TRIDENT

Registros procesados en pequeños lotes:



Ambos enfoques tienen ventajas y desventajas:

	NATIVE STREAMING	MICRO-BATCHING
Latencia	Baja latencia.	Menor latencia en función del intervalo del lote.
Tolerancia a fallos	Menor tolerancia a los fallos.	Mayor tolerancia a los fallos.
Rendimiento	Peor rendimiento. A cada registro se le debe realizar un seguimiento y un punto de control una vez procesados.	Alto rendimiento. El procesamiento y los puntos de control se realizarán de una sola vez para el grupo de registros.
Equilibrio de la carga	Menor equilibrio de la carga.	Mayor equilibrio de carga ya que los datos son distribuidos en menor tamaño.
Operaciones de estado	Operaciones de estado más fáciles.	Operaciones de estado más difíciles.



Entre los servicios actuales de streaming destacan:

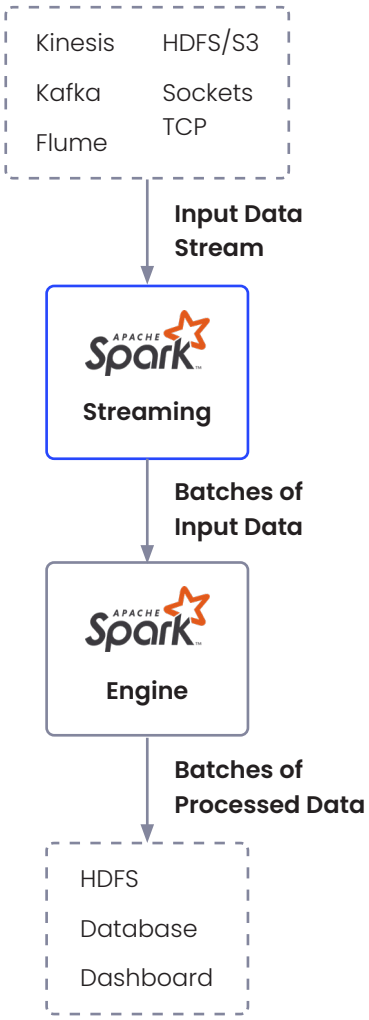
Spark Streaming, basada en Micro-Batch.

Apache Flink, basada en el procesamiento de flujos de baja latencia.

Spark Streaming es una extensión del framework de código abierto Spark que busca dar solución al procesamiento de datos masivo en tiempo real. En líneas generales, convierte un flujo de datos contínuo en uno discreto (discretized stream o DStream) formado por paquetes de datos, para que pueda ser procesado por Spark Core. Es decir, agrupa los datos en micro-batches para poder procesarlos.

Esta herramienta permite un procesamiento en stream de alto rendimiento, escalable y tolerante a fallos. Acepta diferentes fuentes de datos como Kinesis, Kafka o sockets TCP y, tras el procesamiento, los datos pueden enviarse a bases de datos, dashboards en vivo y sistemas de archivos.

Adicionalmente gracias al ecosistema Spark se pueden aplicar algoritmos de machine learning y procesamiento de gráficos a los flujos de datos.

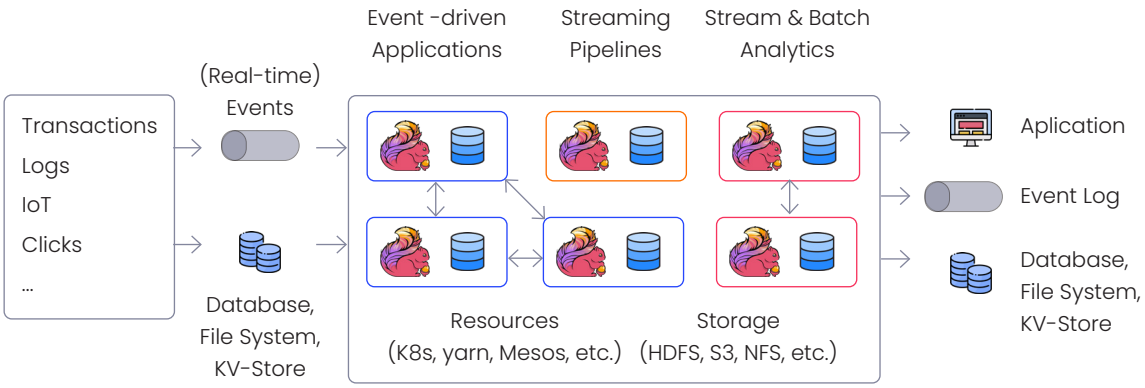




Apache Flink es una plataforma de código abierto, un motor de flujo de datos de streaming que destaca por la comunicación, tolerancia a fallos y distribución. Este framework de análisis de datos escalable es totalmente compatible con Hadoop.

Flink puede procesar transmisiones en vivo en el intervalo de menos de un segundo. Tiene muchos operadores predefinidos para procesar los datos, además, las APIs de Flink se desarrollan de manera que se cubren todas las operaciones comunes, por lo que los programadores pueden usarlas de manera eficiente, facilitando la programación.

Apache Flink proporciona alto rendimiento y baja latencia gracias a su arquitectura sencilla, que es capaz de procesar los datos a gran velocidad y permite una fácil escalabilidad. Incluso, la existencia de un error de hardware, software, nodo o proceso no afecta al clúster. Adicionalmente, cabe destacar que en este servicio la administración de memoria proporciona control sobre la cantidad de esta que se utiliza en determinadas operaciones.





SPARK VS. FLINK

Tanto Flink como Spark se caracterizan por no conservar sus datos en el almacenamiento. Los programadores pueden escribir programas de Big Data con datos de streaming, de tal forma que, al seleccionar datos en cualquier formato, se reducen a pares key–value. A continuación, se pueden ejecutar cálculos en pares adyacentes para producir un valor calculado final. De este modo, pueden conectarse elementos de datos a algoritmos de machine learning para realizar modelos predictivos.

	Apache Spark	Flink
Modelo de cálculo	Micro-Batch.	Basado en operadores.
Motor de streaming	Utiliza micro-batches para todas las cargas de trabajo. Pero no es suficiente para procesar grandes flujos de datos en tiempo real y ofrecer resultados en tiempo real.	Utiliza secuencias para todas las cargas de trabajo: streaming, SQL, micro-batch y Batch.
Optimización	Los trabajos deben optimizarse manualmente.	Viene con un optimizador que es independiente de la interfaz de programación real.
Latencia	Tiene una latencia alta en comparación con Apache Flink.	El tiempo de ejecución de streaming de datos logra baja latencia y alto rendimiento.
Rendimiento	Su procesamiento de flujo no es mucho más eficiente que Apache Flink, ya que utiliza procesamiento de micro-batch.	El rendimiento general es excelente. Utiliza operadores nativos de iteraciones de bucle cerrado, lo que hace que el aprendizaje automático y el procesamiento de gráficos sean más rápidos.
Tolerancia a los fallos	Recupera el trabajo perdido y ofrece una solución sin código ni configuración adicionales.	Se basa en snapshots distribuidas de Chandy-Lamport. Mantiene altas tasas de rendimiento y proporciona fuertes garantías de consistencia a la vez.
Eliminación duplicada	Procesan cada registro exactamente una vez, por lo tanto, eliminan la duplicación.	
Administración de la memoria	Proporciona administración de memoria configurable.	Proporciona administración automática de memoria.
Velocidad	El modelo de procesamiento es más lento que Flink.	Procesa los datos a una gran velocidad.

OTROS PROVEEDORES DE SERVICIOS DE STREAMING

La gran mayoría de proveedores de Cloud ofrecen también sus propias soluciones:



Amazon, a través de **Amazon Kinesis**, ofrece a las organizaciones crear aplicaciones de streaming mediante el editor SQL y bibliotecas Java de código abierto. Kinesis lleva a cabo la carga de la ejecución y el escalado de datos.



Google, por su parte, ofrece herramientas de streaming para la filtración de datos a trasvés de **Cloud DataFlow**.



Microsoft posee **Azure Stream Analytics**, que está diseñado para ofrecer análisis E2E en un corto período de tiempo mediante diferentes lenguajes.



Por último, destacar **StreamSQL** de SQL que está diseñada para crear aplicaciones que manipulen secuencias de datos y realizar vigilancia en tiempo real. Dado que se basa en SQL, es rápido y sobre todo fácil de usar, eliminando así la dependencia de científicos de datos para inspeccionar la información transmitida.

CONCLUSIONES

Apache Spark y Flink son las herramientas de Big Data de próxima generación que captan la atención de los desarrolladores. Ambos marcos proporcionan conectividad nativa con bases de datos Hadoop y NoSQL y pueden procesar datos HDFS.

La principal diferencia entre ellos es que Flink es más rápido que Spark, debido a su arquitectura subyacente. Spark tiene una gran comunidad de usuarios detrás, que ofrecen un apoyo incondicional, no obstante, incluso aunque ya se haya implementado en la producción, Flink presenta una capacidad de streaming mucho mayor, con un buen soporte nativo. Por todo ello, se aconseja elegir Flink en lugar de Spark o una solución de Cloud propia, por su potencia y capacidades.



09. Motores de Búsqueda y Visualización

En cuanto a motores de búsqueda, **el mejor posicionado del mercado es Elastic Stack** ya que no solo sirve para la búsqueda de información, sino que también aporta análisis y visualización de datos, todo ello aportando seguridad.



Elastic Stack

Elastic Stack, también conocido como ELSK, es un conjunto de herramientas Open Source que **permiten recoger datos de cualquier fuente y en diferentes formatos para realizar búsquedas, análisis y visualizarlos en tiempo real.**

Dentro de este conjunto de herramientas cada de una de ellas tiene una función específica:

Interfaz de usuario	Kibana	VISUALIZACIÓN DE DATOS
Almacenamiento, Indexación & Análisis	Elasticsearch	MOTOR DE BÚSQUEDA Y ANÁLISIS
Ingesta	Logstash	PIPELINE DE PROCESAMIENTO DE DATOS
	Beats	AGENTES LIGEROS Y FÁCILES DE DESPLEGAR

Kibana

Kibana permite la **visualización y exploración en tiempo real de grandes volúmenes de datos** mediante la representación gráfica para poder consultar de forma fácil conjuntos de datos complejos.

La herramienta trabaja como la interfaz de usuario gráfica encargada de administrar las herramientas Stack y permite la visualización de la información a través de gráficos. Ofrece gran cantidad de características como el uso de Machine Learning para detectar anomalías, sugerencias inteligentes de visualización de datos gracias a Kibana Lens o el uso de Canvas para la visualización de datos de forma creativa.

Elasticsearch

Elasticsearch es la herramienta principal de Elastic Stack, está construida sobre Apache Lucene y trabaja como un **motor de búsqueda y análisis**.

Es una herramienta que permite el escalado horizontal y es capaz de almacenar datos no estructurados en JSON. Además, se debe destacar su velocidad y su capacidad de visualización de información en tiempo real. Por tanto, esta herramienta es un motor de búsquedas distribuido que permite recuperar documentos JSON en distintos formatos, está basado en Java y facilita una interfaz RESTFUL de acceso.

Logstash

Logtash es un motor para **recoger datos de diversas fuentes, normalizarlos y distribuirlos**. Principalmente, está diseñado para ficheros logs, pero puede leer datos de diferentes tipos de fuentes, como XML, CSV o API-REST, entre otros.

Logstash carga los datos para buscarlos y analizarlos en tiempo real, es decir, permite trasladar la información desde cualquier origen a Elasticsearch.



Beats

Los Beats son **cargadores de datos instalados en servidores que funcionan como agentes de envío de información operacional** a Elasticsearch. En otras palabras, los beats se encargan de recoger las métricas de desempeño de servidores. Existen beats propios de Elastic y otros de código abierto. Hay diferentes tipos de beats para cada tipo de datos:

- **Filebeat**: archivos de log
- **Metricbeat**: métricas
- **Packetbeat**: datos de red
- **Winlogbeat**: logs de eventos de Windows
- **Auditbeat**: información de auditoria
- **Heartbeat**: monitoreo de tiempo de actividad
- **Functionbeat**: agente sin servidor

Elastic ofrece tres soluciones que se desarrollan a partir del Elastic Stack y además permite desplegarlas en cualquier lugar.



SOLUCIONES

Elastic Enterprise Search

Permite la búsqueda de resultados y documentos en workplaces, aplicaciones y sitios web.

- Elastic Workplace Search
- Elastic App Search
- Elastic Site Search

Elastic Observability

Permite la monitorización en tiempo real y centraliza los logs, rastreo de APM y métricas.

- Elastic Uptime
- Elastic Logs
- Elastic APM
- Elastic Metrics

Elastic Security

Permite prevenir, detectar y responder amenazas.

- Elastic Endpoint Security
- Elastic SIEM

CLOUD

SAAS

Elastic Cloud

Servicio alojado y administrado por Elastic.

ORQUESTACIÓN

Elastic Cloud enterprise

Gestión de forma unificada de las soluciones de Elastic.

Elastic Cloud on kubernetes

Orquestación optimizada para kubernetes.



¿CÓMO FUNCIONA?

Beats envía los datos de las máquinas o sistemas a Logstash o Elasticsearch. Logstash permite la recolección de datos a partir de ficheros de los logs, y envía la información para su almacenamiento en el motor de búsqueda y análisis de Elasticsearch. Y, gracias a Kibana se permite la visualización de datos en tiempo real.

Así, Elasticsearch actualiza su información antes de la indexación de datos a través de la ingesta de información de diferentes fuentes y aplicaciones web. Cuando los datos son indexados, los usuarios pueden hacer consultas más complejas, usar distintos medios para recuperar información y analizar los datos de una forma más selectiva. En consecuencia, aparecen los **índices de Elasticsearch**, que son colecciones de documentos que tienen relación entre ellos.

A continuación, esta solución se encarga de almacenar los diferentes datos en documentos JSON. Así el programa emplea una estructura de datos llamada **índice invertido**, diseñado para realizar búsquedas más eficientes. A través de este índice, se pueden realizar listas de palabras únicas que aparecen en los documentos, facilitando su identificación mediante dichas palabras.

Beats

CARGADORES DE DATOS LIGEROS

- Agentes de recolección ligeros.
- Ficheros, métricas, paquetes, eventos de auditoría.
- Módulos preconstruidos para análisis y visualización.



FILEBEAT

Archivos de log



METRICBEAT

Métricas



PACKETBEAT

Datos de red



FUNCTIONBEAT

Agente sin servidor



WINLOGBEAT

Logs de eventos de Windows



AUDITBEAT

Información de auditoria



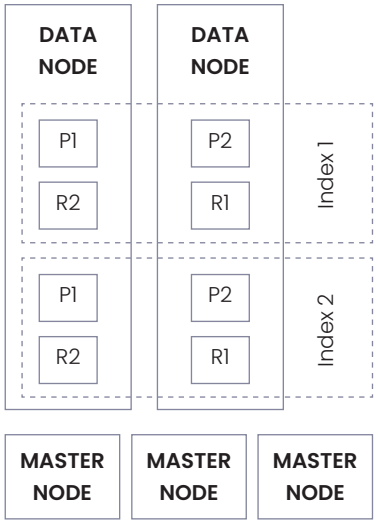
HEARTBEAT

Monitoreo de tiempo de actividad

Elasticsearch

MOTOR DE BÚSQUEDA Y ANÁLISIS

- Buen rendimiento en búsquedas y analítica.
- Escalable, resiliente, alta disponibilidad.
- Tipos de nodos configurables.



Logstash

PIPELINE DE PROCESAMIENTO DE DATOS

- Normalizar, filtrar, enriquecer.
- Configuración centralizada.
- Colas persistentes.



INPUTS

Beats, udp, tcp, http, jdbc, kafka, s3, redis, syslog, etc.



FILTROS

Grok, mutate, date, fingerprint, geoip, useragent, json, wml, etc.



OUTPUTS

elasticsearch, cloudwatch, graphite, Kafka, s3, etc.

Kibana

VISUALIZACIÓN Y GESTIÓN

- Explorar y buscar.
- Interacción visual con los datos.
- Herramientas de desarrollo y gestión.



DESCUBRIMIENTO

Búsqueda y filtrado de datos de elasticsearch en tiempo real.



VISUALIZACIÓN

Creación de gráficos, tablas de datos, diagramas, mapas, etc.



DASHBOARD

Búsqueda y visualización en un único panel.



GESTIÓN

Solución de problemas, ajustes, configuración, etc.

CASOS DE USO

Entre sus casos de uso destacan principalmente el uso **para desarrollo de software y DevOps**, que a través del panel de control de Kibana se obtiene una mayor visibilidad de la aplicación y el servidor donde se emplee.

Otro uso común es el **soporte de sistemas**. Debido al alto nivel de monitorización que ofrece Stack, se pueden controlar y gestionar los logs de las aplicaciones y, en consecuencia, detectar los errores de estas y gestionarlas con anticipación. Generalmente, también se usa para redes, debido a la facilidad de acceso a la monitorización de interfaces de red.

Existen diversos casos de uso de Elastic Stack y, normalmente, están documentados en la página oficial de Elastic (como Netflix, Uber o Slack), pero existen algunos que no se recogen en su web y se dan a conocer mediante blogs o a través de la comunidad. A continuación, se mencionan varias **aplicaciones conocidas que hacen uso cotidiano de Elastic:**

Uber

UBER: La aplicación de este servicio de movilidad permite su seguimiento en tiempo real o la búsqueda de tarifas dinámicas a través de los mapas de calor gracias a Elasticsearch. La información recopilada con la aplicación trabaja en tiempo real.

tinder

TINDER: La aplicación de citas más conocida cruza los intereses comunes de dos individuos a través del motor de búsqueda que implica Elasticsearch, pero no solo los intereses, sino también la ubicación y diversas variables más para entregar en tiempo real.

LinkedIn

LINKEDIN: Esta plataforma de marketing y búsqueda de trabajo usa Elastic Stack para conocer su rendimiento y monitorizar su seguridad. El equipo de LinkedIn instaló ELK junto con Kafka para soportar la carga en tiempo real. La aplicación de LinkedIn incluye cientos de clústeres en al menos seis centros de datos diferentes.



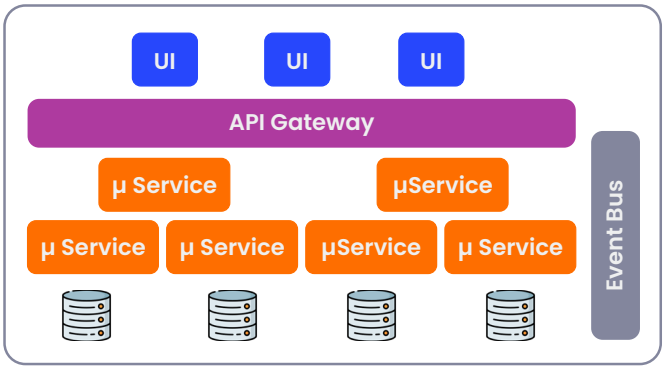


10. Gestión de Eventos

Cuando los equipos trabajan en microservicios que serán implementados de manera independiente, una forma de **que el sistema general tenga coherencia** es trabajar con una arquitectura dirigida por eventos.

Una **arquitectura dirigida por eventos** (EDA, Event-driven architecture) es un patrón de diseño que permite a un conjunto de sistemas comunicarse entre sí de forma reactiva, mediante la publicación y el consumo de eventos. Estos, son sucesos o cambios en el estado de un elemento del sistema, y pueden originarse por estímulos internos (cuando se ha terminado un procesamiento) o externos (la actividad de un usuario).

Event Driven Architecture

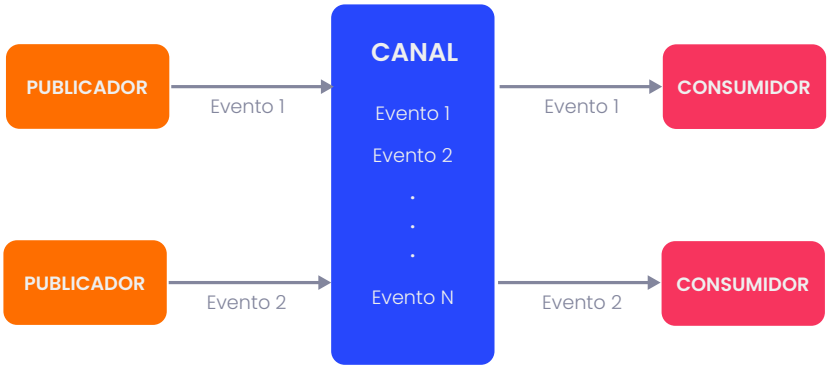




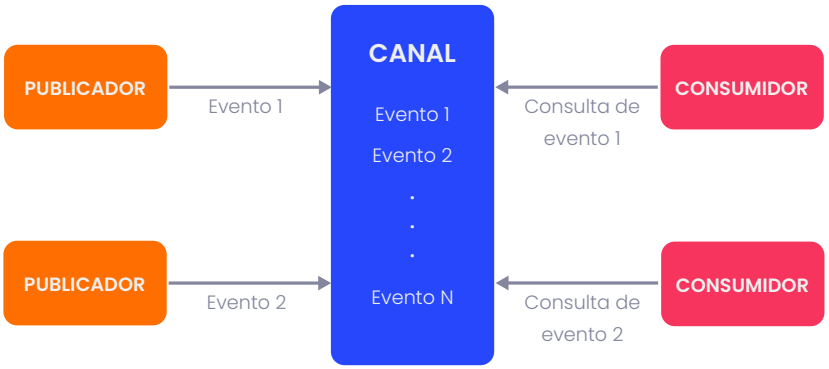
Es una arquitectura con un bajo nivel de acoplamiento, lo que la convierte en una buena opción para arquitecturas de aplicaciones distribuidas modernas. Este patrón convierte en una realidad la creación de arquitecturas con respuesta en tiempo real.

Una arquitectura dirigida por eventos está compuesta por **consumidores y productores** de eventos, y la relación entre ellos puede estar basada tanto en modelo publ/sub como en un modelo de event streaming (flujo de eventos):

Modelo publ/sub



Modelo event streaming



LAS PRINCIPALES VENTAJAS DE UNA EDA SON:



Es **asíncrona**, y por tanto sin bloqueos y con alto rendimiento.



Una EDA con una cola puede **recuperar trabajo perdido** al "repetir" eventos del pasado.



Los **servicios están desacoplados**, no necesitan saber o depender de otros servicios.



El **escalado resulta muy sencillo**, ya que cada procesador de eventos puede escalar por separado.



kafka

La plataforma de uso más conocida para la transmisión de eventos es Apache Kafka. Fue desarrollado por LinkedIn y, actualmente, es un proyecto open source mantenido por Confluent, que pertenece a Apache. Se define como una **plataforma de streaming de eventos distribuidos para pipelines de datos de alto rendimiento, análisis de streaming, integración de datos y aplicaciones de misión crítica**.

Fue diseñado para solventar la necesidad de arquitecturas de alto rendimiento y escalables, que pudiesen almacenar, procesar y reprocesar datos en streaming.

POSEE TRES FUNCIONALIDADES CLAVE:

LA PUBLICACIÓN Y SUSCRIPCIÓN A FLUJOS DE DATOS como un sistema de cola de mensajes, pero ofreciendo mayor rendimiento a menores latencias en relación con la transmisión de mensajes. También, permite dividir el procesamiento de datos en múltiples instancias de consumidores permitiendo escalar su procesamiento.

ALMACENA STREAMS Y SU RÉPLICA para ofrecer corrección de fallos. Asimismo, Kafka indica el estado del proceso para que una escritura no se considere completa hasta que no está replicada, garantizando así su existencia.

FACILITA EL PROCESAMIENTO DE STREAMS transformando así los datos que se almacenan en la herramienta.

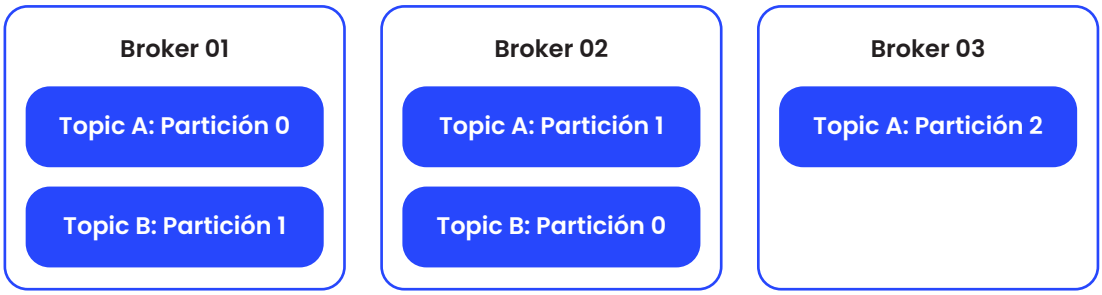
ARQUITECTURA DE KAFKA

Apache Kafka permite una **comunicación asíncrona** que evita los cuellos de botella que existen en las arquitecturas monolíticas con bases de datos relacionales. Además, Kafka también evita las interrupciones debido a su **alta disponibilidad**, y en caso de que exista una falla, la interrupción del servicio es mínima. Paralelamente, su arquitectura permite la opción de volver a cierta información y reproducir eventos si son necesarios, gracias a la **retención de datos** durante un periodo de tiempo configurado.

Los principales componentes que definen la arquitectura de Kafka son:

BROKERS

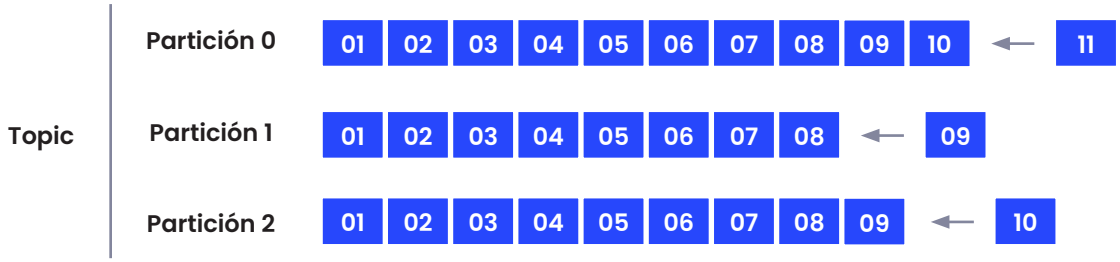
Los brokers de Kafka son los **servidores donde se almacenan uno o más clusters**. Cada broker tiene una identificación ID que contienen particiones de un topic.



TOPIC

Un topic es un **flujo de datos sobre un tema**. Kafka permite crear tantos Topics como se deseen y, a su vez, estos pueden dividirse en particiones.

Asimismo, cada elemento se almacena en un topic, llamado **mensaje**. Estos mensajes se añaden a una partición determinada en el orden en el que se envían, de tal forma que existe orden dentro de una partición, pero no tiene por qué existir entre ellas.





Cada partición estará siempre identificada por un numero incremental llamado **offset**. Así pues, los mensajes se guardan en los topics durante tiempo limitado, pero el offset sigue creciendo numéricamente.

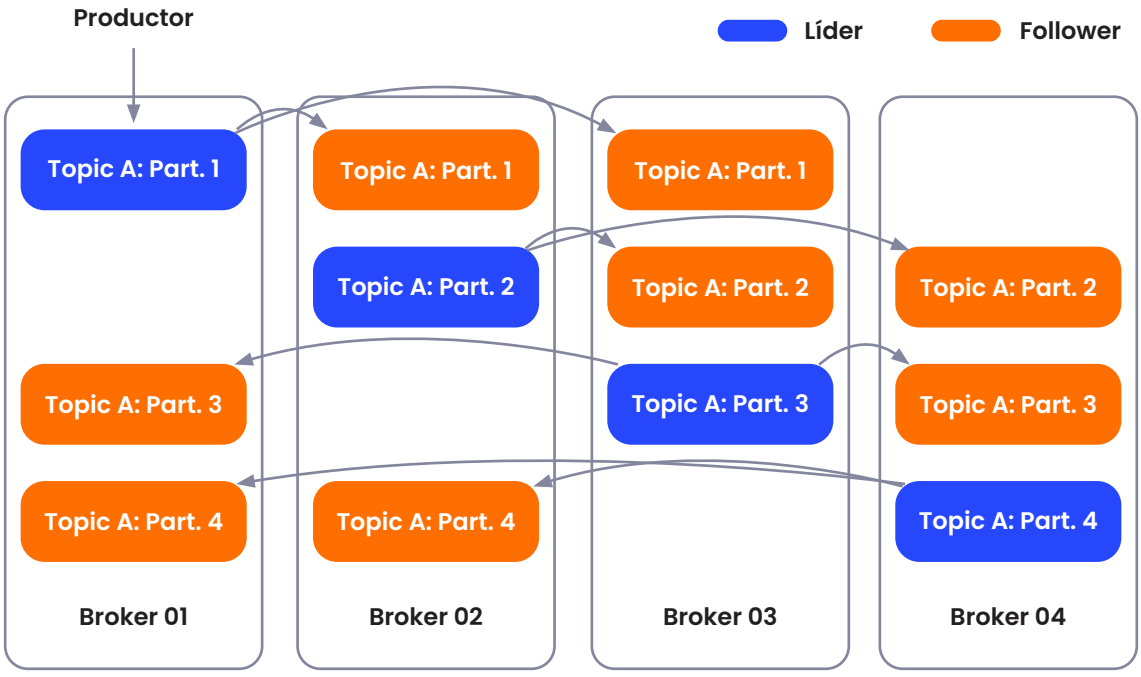
Los topics son **multi-productores y multi-consumidores**, lo que significa que cualquier fuente puede enviar datos a un topic (productor), y cualquier fuente puede leer los datos (consumidor).

- **Topic replication**

Los brokers permiten replicar los topics mediante un balanceo de cargas de almacenamiento, y es esto lo que define su escalabilidad y tolerancia a los fallos.

Normalmente, los topics en Kafka tienen un factor de replicación de 2 o 3, de forma que, **si un broker se cae, otro proporciona los datos**.

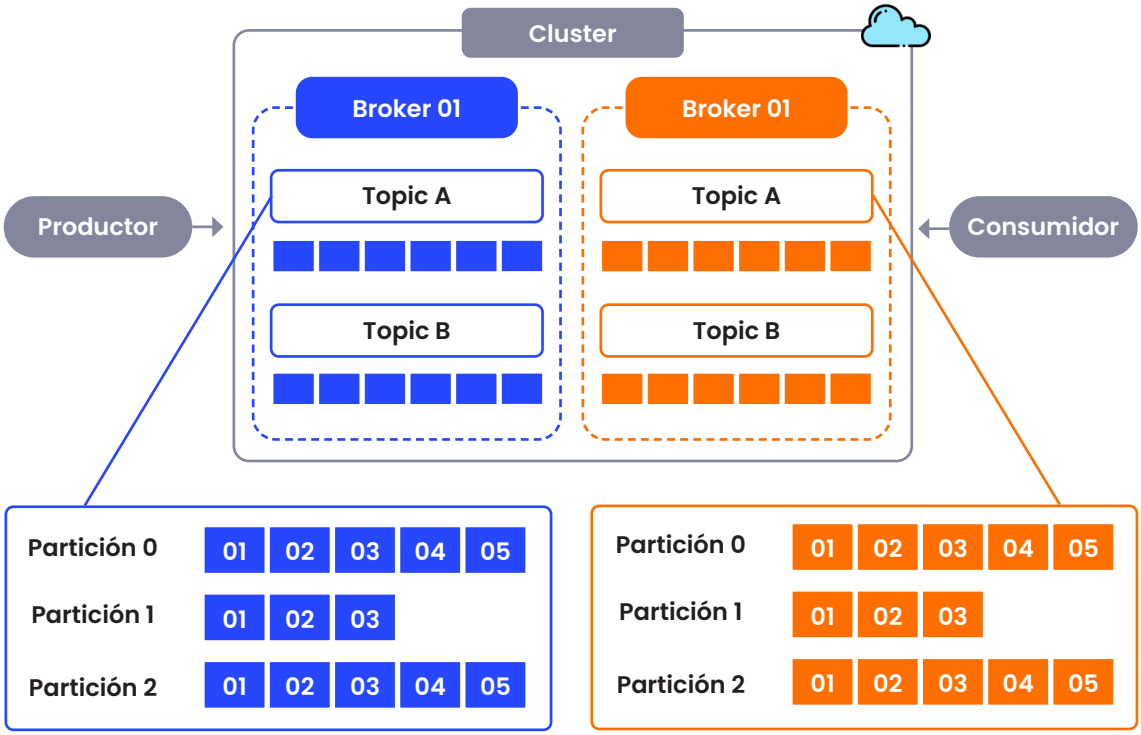
Únicamente puede existir un **broker líder** para cada partición de un topic, y será este el que recibe y proporciona datos de una partición, mientras que el resto de brokers sincronizan sus datos.



ZOOKEEPER

Zookeeper es un **servicio centralizado donde se envían las notificaciones en caso de cambios** como la creación de un nuevo topic, la caída de un bróker o borrado de topic, entre otros.

Por tanto, su función principal es la gestión de los brokers mediante un listado de sus metadatos y su análisis. Paralelamente, realiza la selección de un bróker líder para las diferentes particiones de los topics.



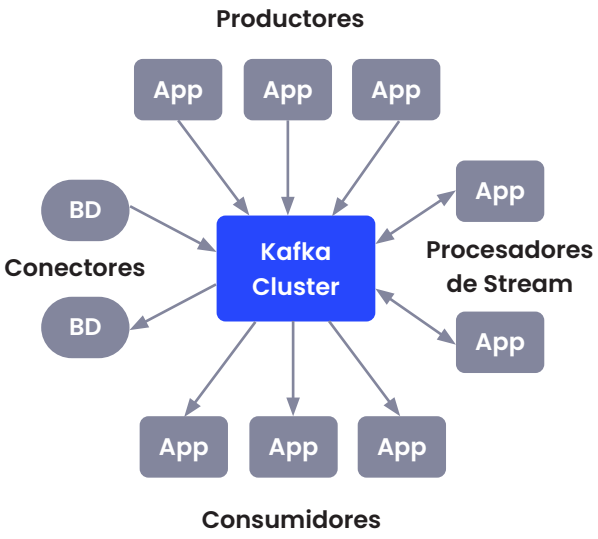


Kafka tiene cinco APIs clave que permiten procesar eventos de streaming:

Producer API: envía flujos de datos a topics del cluster de Kafka.

Consumer API: lee flujos de datos desde los topics del cluster de Kafka.

Streams API: transforma los flujos de datos de topics de entrada a topics de salida.



Connect API: implementan conectores que de forma constante extraen de algún sistema o aplicación de origen a Kafka o introducen desde Kafka a algún sistema o aplicación de destino.

Admin API: gestiona e inspecciona topics, brokers u otros objetos de Kafka.

CONCLUSIÓN

Actualmente, entre las diferentes soluciones del mercado destaca Kafka, capaz de combinar las ventajas de un servicio publicación-suscripción junto con una cola de mensajería. Esta plataforma puede ofrecer millones de eventos al día gracias a su escalablilidad y gran rendimiento.

Esto hace a Apache Kafka una alternativa idónea para construir soluciones de procesamiento de eventos en tiempo real y una de las opciones preferidas para los entornos distribuidos. Además, **Confluent brinda una plataforma Enterprise que amplía la potencia de Kafka aportándole nuevas capacidades** como REST, conectores, esquemas, etc.



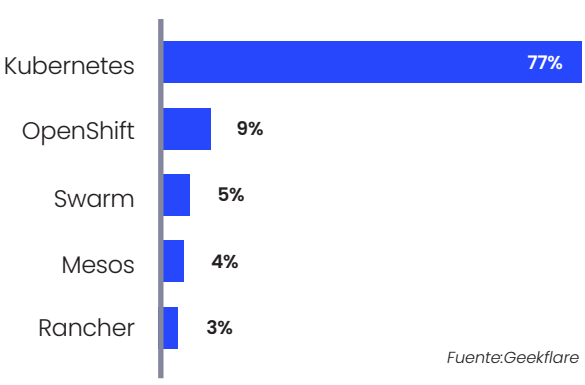
11. Contenedores

Con el crecimiento exponencial de aplicaciones y sus actualizaciones, hoy en día, la gestión en contenedores se hace cada vez más difícil.

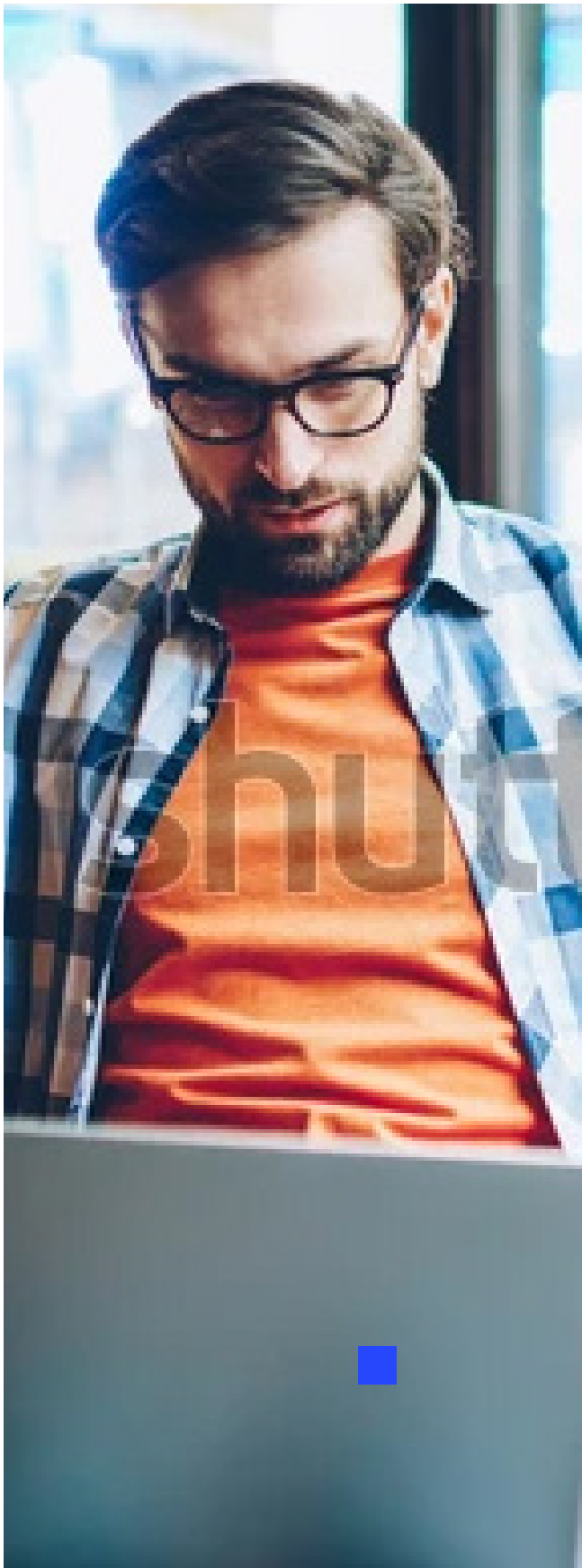
La principal solución actual a esta problemática es Kubernetes, cuya finalidad es proporcionar una API de código abierto que controle la forma y el lugar donde se ejecutan los contenedores.

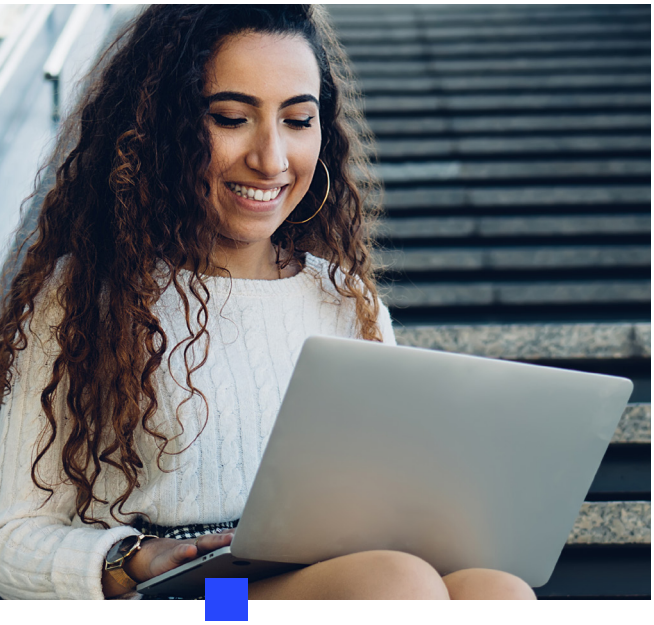
Es el resultado del esfuerzo de múltiples partes interesadas que lo han desarrollado bajo la supervisión de la Cloud Native Computing Foundation (CNCF) de la fundación Linux. Con ello, **Kubernetes facilita la automatización de aplicaciones** y abarca una gran comunidad de usuarios, en constante crecimiento. Su cuota de mercado en la orquestación de contenedores es indiscutible: un 77%, frente a alternativas minoritarias como OpenShift o Swarm.

Cuota de mercado de herramientas de orquestación de contenedores



Fuente:Geekflare

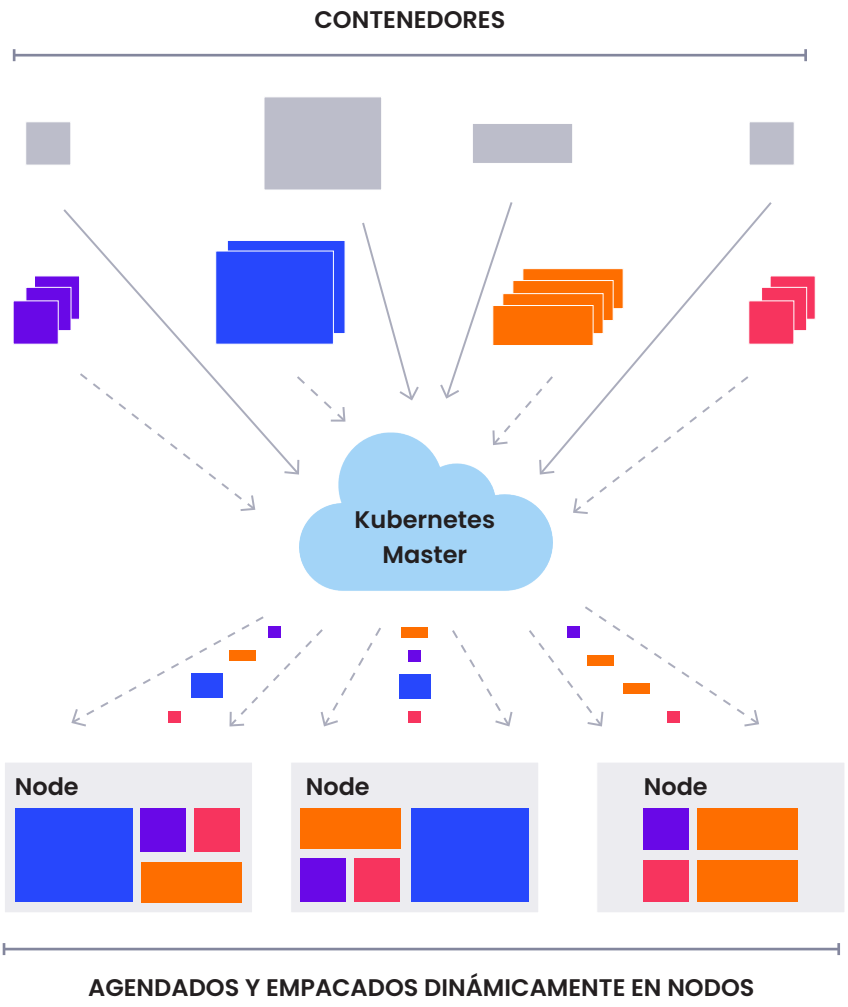




kubernetes

La metodología de trabajo de Kubernetes es la **formación de clústeres mientras se programan contenedores**, para que se ejecuten las aplicaciones en dichos clústeres de máquinas virtuales en función de los recursos de procesamiento necesarios y los requisitos de cada contenedor.

Por otra parte, también permite administrar los servicios y escalarlos automáticamente. Además, los contenedores se pueden replicar con facilidad, y comprobar su estado de manera automática.



BENEFICIOS DE USAR KUBERNETES



La solución de Kubernetes para la gestión de muchos contenedores en un mismo sistema pasa por **contenedores que se programan e implementan a la par que se escalan al estado deseado**.



De este modo, cuando las aplicaciones se ejecutan en Kubernetes **dejan de estar vinculadas a la infraestructura y se convierten en portables**. Por tanto, su gestión se puede llevar a cabo en un entorno local, híbrido o multi-Cloud, de forma que se mantiene la coherencia entre los distintos entornos.



Finalmente, **permite crear aplicaciones más extensibles**. La CNCF fomenta la creación de extensiones y complementos que aportan funcionalidad a Kubernetes.

FUNCIONALIDADES Y USOS

Kubernetes tiene un **amplio abanico de funcionalidades y usos**. Por ejemplo, permite agregar volúmenes persistentes a los nodos de trabajo. Estos nodos se basan en discos Classic o High Speed, con facturación por gigabytes, garantizando así la durabilidad de los datos en las aplicaciones stateful. Además, Load Balancer permite la gestión óptima de los nodos en función de su carga.

KUBERNETES TIENE TRES GRANDES USOS:



Puede funcionar mediante aplicaciones legacy para repartir tareas en arquitecturas de microservicios con el fin de reducir las infraestructuras subyacentes.



Permite actualizaciones de versiones transparentes y controladas, al optar por el enfoque del rolling upgrade.



La gestión de contenedores entre los nodos de Kubernetes permite **aislar desarrollo, preproducción y producción**, trasladando los archivos de configuración de un clúster a otro, de un modo sencillo y rápido.

En 2021 veremos incluso una mayor adopción de Kubernetes, ya que es el componente principal de la estrategia Hybrid-Cloud o Multi-Cloud. Incluso aplicaciones de AI/ML o bases de datos también lo utilizarán como plataforma.

SERVICIOS DISPONIBLES EN EL MERCADO

Casi todas las empresas de **Giant Tech ofrecen un servicio gestionado por Kubernetes** junto con sus servicios de contenedorización:

Microsoft Azure Kubernetes Service (Azure AKS) ofrece Kubernetes, con una experiencia de integración y entrega continuas (CI/CD). A través de KEDA, AKS proporciona un aprovisionamiento elástico sin necesidad de administrar la infraestructura junto con la escalabilidad automática basada en eventos. También aporta una experiencia de desarrollo integral más rápida y una administración avanzada del acceso a través de Azure Active Directory, así como una implementación de reglas dinámicas en varios clústeres.

Google Kubernetes Engine (Google GKE) proporciona un servicio de contenedores mediante la infraestructura de Google. En este caso, Google Cloud administra los clústeres como un balanceo de cargas cuando dicho clúster se ejecuta. También cuenta con un ajuste automático para el escalado de nodos; y actualizaciones, reparaciones y supervisiones automáticas de los nodos y clústeres con Google Cloud's Operations Suite.

Amazon Elastic Kubernetes Service (Amazon EKS) permite trabajar aplicaciones de Kubernetes en la nube de AWS o a nivel local. Amazon EKS es capaz de automatizar diversas tareas como la gestión y actualización de los nodos, proporcionando clústeres seguros. Además, no es necesario refactorizar el código para migrar cualquier aplicación de Kubernetes, ejecutando Kubernetes upstream. Asimismo, se puede disponer de una distribución probada de código abierto de Kubernetes donde se quiera para trabajar con Amazon EKS Distro.





EL NACIMIENTO DE CONTAINER RUNTIME INTERFACE (CRI)

Docker convirtió el uso de los contenedores en algo tan simple y sencillo, que rápidamente pasó a ser el método estándar en el que las personas ejecutaban dichos contenedores. Pero actualmente, Docker está siendo desaprobado por Kubernetes, ya que presenta un tiempo de ejecución considerablemente más lento que el de Containerd y CRI-O.

Kubernetes está mostrando su apoyo a los tiempos de ejecución que utiliza Container Runtime Interface (CRI), hasta tal punto que planea eliminar a Docker completamente a finales de 2021 en su próxima versión (1.22).

CRI es una API que permite no modificar la base del código cada vez que exista un nuevo tiempo de ejecución del contenedor. Esta API se encarga de definir los tiempos de ejecución de cada contenedor, evitando así la responsabilidad de Kubernetes.

En resumen, **la interfaz CRI es la que en última instancia ejecuta los contenedores en Kubernetes**. Actualmente, CRI-O, Fratki, CRI-Containerd y Dockershim son los tiempos de ejecución de contenedores que usa la API CRI, que están reemplazando a Dockers.



12. Modelos de diseño de APIs

La arquitectura de microservicios está cambiando hacia una comunicación síncrona entre eventos y para ello son necesarias las APIS. Una API es una interfaz que establece la forma de comunicación entre softwares, a través de programas que instauran funciones predefinidas y normas para dicha comunicación. Dicho de otro modo, las APIs son un conjunto de **protocolos que se integran en los softwares para permitir la comunicación entre ellas**.

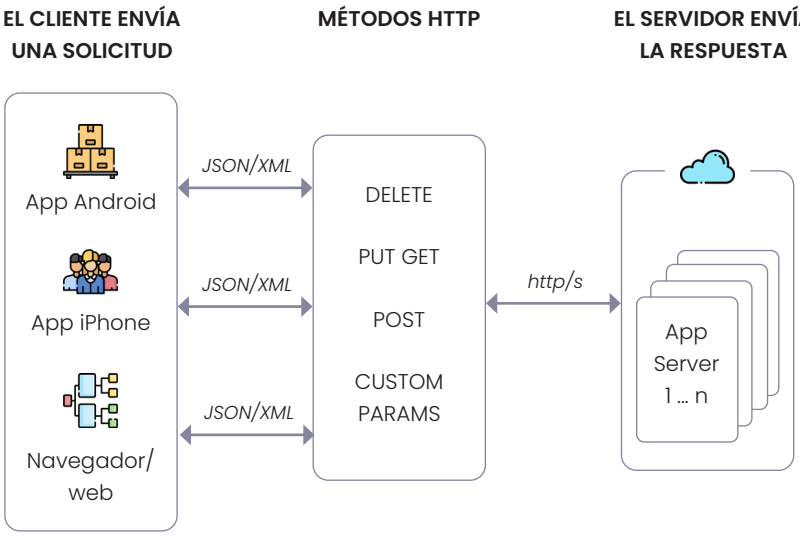
Actualmente, existen tres modelos request/response principales para el diseño de las APIs que son REST, gRPC y GraphQL. Independientemente del modelo, la mayoría de las API modernas se implementan mapeándolas al mismo protocolo HTTP.



REST

REST ha sido el protocolo API más dominante en 2020. Se basa en un protocolo de transporte estándar, HTTP, para intercambiar información entre diferentes aplicaciones o servicios. Este protocolo **se caracteriza por ser muy estable y estar en una plataforma unificada**.

REST API Architecture



HTTP está intrínsecamente orientado a la entidad; es una forma de lograr una estructura unificada, esto es, modelar todos los comportamientos mediante un conjunto estándar de procedimientos para cada tipo de entidad.

REST es un modelo orientado a entidades que **aporta una organización general para los comportamientos del sistema** como, por ejemplo, el modelo de entidad de compras on-line, con productos, carritos, pedidos y cuentas. Los modelos orientados a entidades son más simples, más regulares, más fáciles de entender y más estables con el tiempo que los modelos RPC simples.

CASO DE USO

PayPal

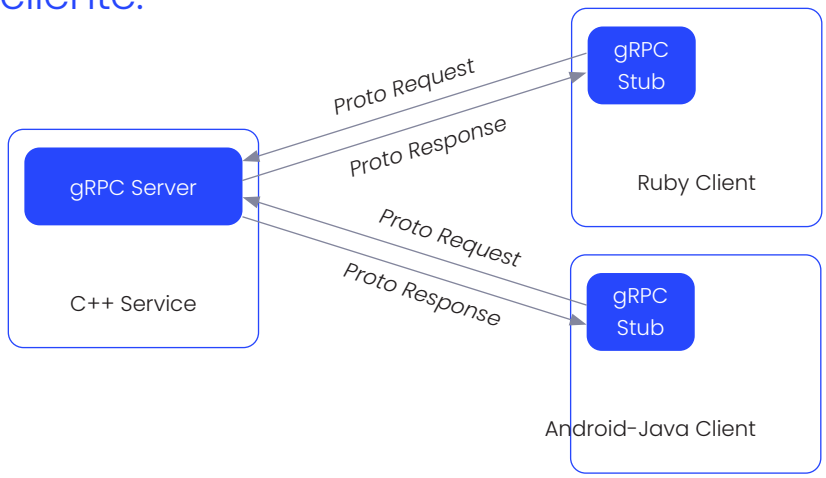
La API de PayPal se basa en REST porque debe ser capaz de **facilitar sistemas integrados para el procesamiento de pagos, de tal forma que los recursos se puedan identificar fácilmente**. La comunicación debe admitir diferentes medios para controlar y gestionar de manera eficiente los diferentes métodos de pago.

gRPC



gRPC es un protocolo RPC (Remote Procedure Call) para implementar las APIs. Funciona en un modelo en el que las entidades direccionables son procedimientos, y los datos se ocultan detrás de estos procedimientos. Este protocolo utiliza **Protocol Buffers** como formato de mensajería, idóneo en la comunicación machine-to-machine.

Comunicación entre el servidor gRPC y la App del cliente:



gRPC **proporciona diferentes marcos de trabajo, así como una variedad de bibliotecas** fáciles de implementar. Estas bibliotecas pueden ser muy útiles porque son intuitivas y fáciles de ejecutar.

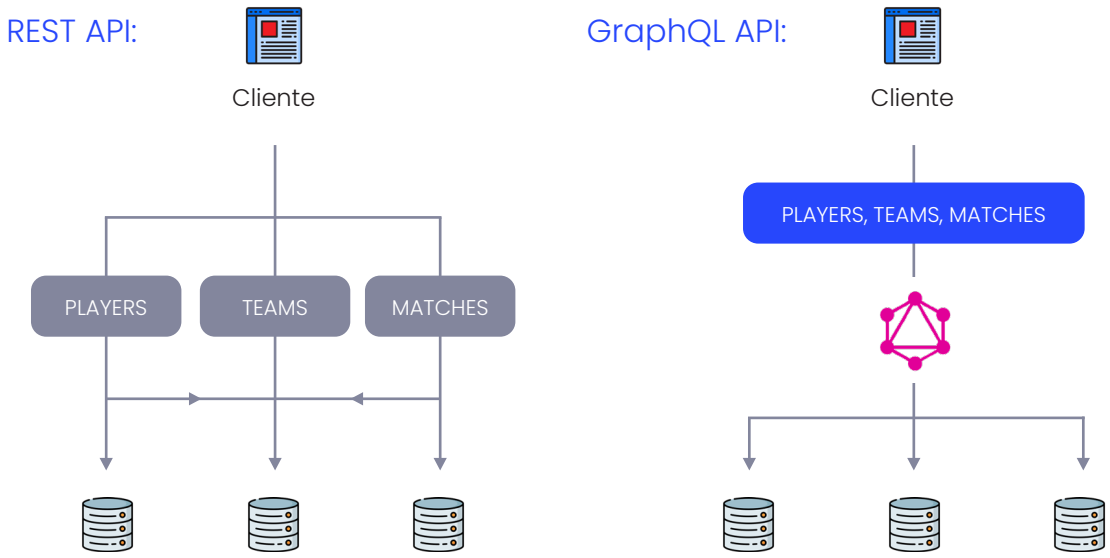
A diferencia de REST, gRPC expresa una API RPC en un **lenguaje de descripción de interfaz (IDL)**. El IDL de gRPC proporciona una forma más sencilla de definir procedimientos remotos. Además, este protocolo hace uso de una carga binaria para crear y analizar las conexiones, en vez de necesitar softwares especiales en el servidor, por lo que el rendimiento se optimiza.

La funcionalidad principal de gRPC es ser el intermediario del procesamiento de datos entre las APIs de Google Cloud, gracias a sus mecanismos de trasporte estándar y a la carga de datos ágil. De esta forma, gRPC genera comunicaciones totalmente optimizadas.

GraphQL

GraphQL es un lenguaje de consulta y manipulación de datos para las APIs. **Es capaz de reducir la comunicación entre la UI y el back-end**, que habitualmente da lugar a errores cuando la UI requiere comunicación constante. Con este protocolo se optimizan los datos solicitados, ofreciendo un servicio altamente escalable, con mayor flexibilidad y disponibilidad.

Su principal ventaja es la **capacidad de desarrollar un sitio web único**, ya que separa la parte back-end de cualquier proyecto. Dicho proyecto es capaz de funcionar en diferentes plataformas, definiendo la información a recibir y su formato, siendo más sencillo el desarrollo front-end.



GraphQL puede ser combinado con ubicaciones web ya existentes que implementen REST, es decir, que **su uso no es excluyente e incluso ayuda a reducir la complejidad de los sitios**.

Fundamentalmente, al reducir la dificultad de programar el back-end, se reduce la posibilidad de que aparezcan errores, así como la reducción de solicitudes para visualizar datos pesados. Esto implica que **solo sea necesario gestionar una única versión de la API**.

CASO DE USO

GitHub

GitHub hace uso de esta API evitando múltiples llamadas a datos complejos, permitiendo la transformación de solicitudes de los usuarios simplificando el requerimiento. A su vez, es capaz de llevar a cabo la gestión de solicitudes básicas.



CONCLUSIÓN

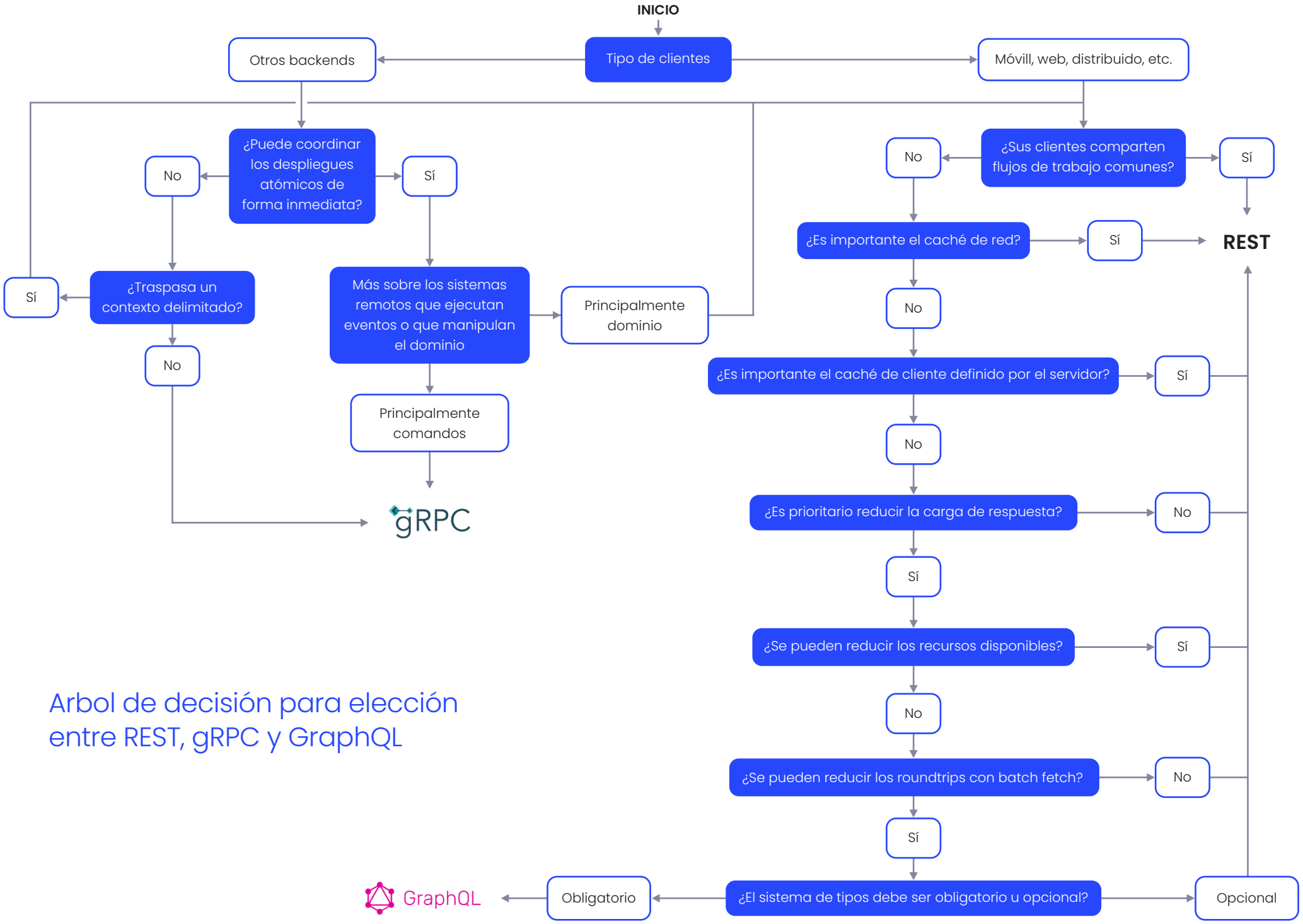
La elección de una API se realiza en función de su uso especifico teniendo en cuenta cuál de ellas se adapta mejor al código base. Las principales APIs de uso en el mercado son REST, gRPC y GraphQL.

REST es la dominante en el sector por su arquitectura específica para la trasferencia de datos, gRPC presenta un sistema ágil para la solicitud de datos, y en GraphQL se define el formato de los datos a través del usuario.

Lo más correcto es decir que algunas APIs son mejores para realizar unas funciones básicas en función del proceso que se vaya a desarrollar.

Principales diferencias entre REST, gRPC y GraphQL:

	REST	gRPC	GraphQL
Arquitectura	Muy conocida y popular.	Está ganando popularidad.	Poco conocida.
Comunicación	Comunicación sincrona sólo en protocolo HTTP.	Comunicación síncrona/ asincrona en múltiples protocolos como HTTP, AMQP o MQTT.	Comunicación asíncrona en protocolo HTTP/2.
Diseño	Diseño totalmente basado en HTTP (Status, Methods y URI).	Diseño basado en cambio de mensajes.	Diseño basado en cambio de mensajes.
Estándares de especificación	Usa múltiples estándares de especificación (OpenAPI, RAML, etc.).	Usa como estándar de especificación Schema Definition (IDL).	Usa como estándar de especificación protobuf (IDL).
Estándar de tramas	El estándar de tramas es JSON.	El estándar de tramas es JSON.	El estándar de tramas es serializado (protobuf).



Arbol de decisión para elección entre REST, gRPC y GraphQL



13. DL Library

El aprendizaje automático implica el procesamiento de una gran cantidad de datos por lo que se hace indispensable utilizar librerías potentes para llevarlo a cabo. Dos de las más importantes son TensorFlow 2.0 y Pytorch, ambas son **librerías de computación numérica, orientadas a resolver problemas y desarrollar algoritmos de Deep Learming**.

Estas librerías son dos de los marcos más populares que se utilizan hoy en día para construir y optimizar redes neuronales. TensorFlow 2.0 está respaldado y diseñado por Google mientras que Pytorch es creación de Facebook.

TensorFlow

TensorFlow constituye un ecosistema de recursos de código abierto destinado a resolver problemas complejos a través de Machine Learning. Es una librería orientada a construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

El impacto que TensorFlow ha tenido en la comunidad ha sido enorme, **convirtiéndose en el soporte sobre el que multitud de nuevos productos innovadores se apoyan**.

Técnicamente es compleja, por lo que para iniciarse en el diseño de redes neuronales y en la creación de modelos es recomendable utilizar Keras ya que esta librería es más simple y se ejecuta sobre TensorFlow.

Keras permite crear prototipos sencillos y de manera eficaz ya que tiene muchas capas incorporadas, evitando, de esta manera, que sea tedioso codificarlos desde cero.

CASOS DE USO

Desde startups de Silicon Valley hasta grandes corporaciones como Airbus, IBM, Twitter o Uber, están usando TensorFlow en sus proyectos.

AIRBUS IBM Uber

EXISTEN TRES MANERAS DE CREAR UN MODELO DE RED NEURONAL EN KERAS:



API funcional: En la API funcional se puede crear una instancia. Es una forma fácil de construir una red neuronal y se recomienda frente a la subclasificación de modelos. Con este enfoque, esencialmente se define una capa e inmediatamente le pasa la entrada de la capa anterior. Por lo tanto, requiere menor codificación con el mismo resultado.



API de modelo secuencial: Permite definir un modelo completo para ciertas redes neuronales (simples), que normalmente consisten en unas pocas capas. Es adecuada y funciona correctamente, pero es demasiado inflexible si se desea implementar ideas más sofisticadas.



Subclases de modelos: Se pueden crear modelos totalmente personalizables Keras usando el método Call. El beneficio de utilizar un enfoque orientado a objetos de este tipo es que se puede reutilizar varias capas dentro del método Call o definir un paso hacia delante más complejo.

Una de las fortalezas de TensorFlow 2.0 es que se puede compilar manualmente el modelo pasando un conjunto de tensores de output y otro conjunto de tensores de input, aunque el modo más sencillo de hacerlo es a través de una **Eager Execution**, que evalúa las operaciones inmediatamente. De este modo, TensorFlow utiliza el paradigma Eager Execution que permite ejecutar directamente para evaluar operaciones de inmediato sin necesidad de crear un gráfico, lo cual permite depurar modelos de forma efectiva, reduciendo los archivos repetitivos. **Las principales ventajas de Eager Execution son**:



Debugging facilitado



Menos código boilerplate



Desarrollo más intuitivo



Fácil de aprender

Las aplicaciones de esta herramienta repercuten en un innumerable abanico de campos empresariales y académicos. Algunas de las ventajas competitivas de TensorFlow son:



Enorme base de usuarios.



Incluye los algoritmos de ML más populares.



Formación ofertada.



Está integrado con los datos de Google Research.



Permite comprimir y optimizar modelos para dispositivos de IoT con TensorFlow Lite



Herramientas y plataformas que ayudan al desarrollo de aplicaciones de ML.

PyTorch

PyTorch, similar a TensorFlow, se emplea principalmente para la investigación, más que para producción, pero no deja de emplear un código sencillo potencialmente útil para las empresas. De hecho, **se creó para hacer más eficiente el desarrollo de modelos**, y es cada vez más utilizado por profesionales del desarrollo debido a su sencillez.

A diferencia de TensorFlow, PyTorch emplea **módulos de lenguaje C++ para la autodiferenciación**, que se encarga de calcular automáticamente el gradiente de las funciones definidas durante la retropropagación. También utiliza el cálculo de eager model de manera predeterminada. De este modo, puede ejecutar una red neuronal a medida que la compila, lo que facilita la depuración. También permite construir redes neuronales con ejecución condicional, siendo más dinámica e intuitiva para la mayoría de los programadores de Python.

PyTorch se basa en **Torch**, un marco de cálculo rápido que está escrito en C. PyTorch ajusta C backend en una interfaz de Python, pero es más que un wrapper. El código subyacente de C y C++ está optimizado para ejecutar código Python. Debido a esta estrecha integración, se obtiene:



Mejor memoria y optimización.



Mensajes de error más concretos.



Control más preciso de la estructura del modelo.



Comportamiento del modelo más transparente.



Mejor compatibilidad con NumPy.

En conclusión, se pueden escribir componentes de red neuronal altamente personalizados directamente en Python sin tener que utilizar muchas funciones de bajo nivel. De hecho, la Eager Execution de PyTorch, que evalúa las operaciones de tensor de forma inmediata y dinámica, inspiró TensorFlow 2.0, por lo que las API de ambas herramientas tienen grandes similitudes.





Comparativa Tensor Flow 2.0 vs PyTorch:

TensorFlow	PyTorch
No es necesario inicializar sesiones ya que sólo se utilizan funciones.	PyTorch está estrechamente integrado con Python.
La API de Keras, que también es una API de alto nivel, se utiliza para la implementación.	La API de REST se utiliza junto con Flask para la implementación.
La API de TensorFlow es más difícil de depurar debido a su enfoque de gráfico de cálculo estático, aunque Keras ayuda a resolver esta dificultad.	La API de PyTorch es más sencilla dado que el lenguaje de programación es similar a Python nativo.
Tiene ventaja sobre PyTorch en el aspecto de la implementación debido a herramientas que la facilitan a través de la nube.	PyTorch ha tratado de cerrar la brecha en la versión 1.5+ con TorchServe pero aún sigue sin ofrecer resultados similares a los de TensorFlow 2.0.
Pueden ejecutarse en CPU o GPU individuales, múltiples o distribuidas.	
Ambos admiten gráficos de cálculo estáticos y dinámicos.	



CONCLUSIONES

En conclusión, TensorFlow y PyTorch son dos marcos excelentes para la investigación y el desarrollo de aplicaciones de Deep Learning. Gestionan el hardware subyacente y tienen un gran acceso a los datos, lo que facilita desarrollar aplicaciones totalmente personalizadas. En cualquier caso, parece obvio que **TensorFlow 2.0 y PyTorch serán bibliotecas "go-to" para empresas y comunidades**.

Para las empresas con una base de código TensorFlow existente o que quieran usar Collab Notebook se recomienda usar TensorFlow 2.0, mientras que para las empresas o individuos más inexpertos o que deseen continuar con Juypter Notebook, se recomienda elegir PyTorch.

Aun así, existen **otros frameworks** como: Caffe, para la clasificación de imágenes; Microsoft CNTK; o Theano, para trabajar con arrays multidimensionales.

Microsoft Cognitive Toolkit

theano

Caffe




Softtek®


vector ITC
a Softtek Company