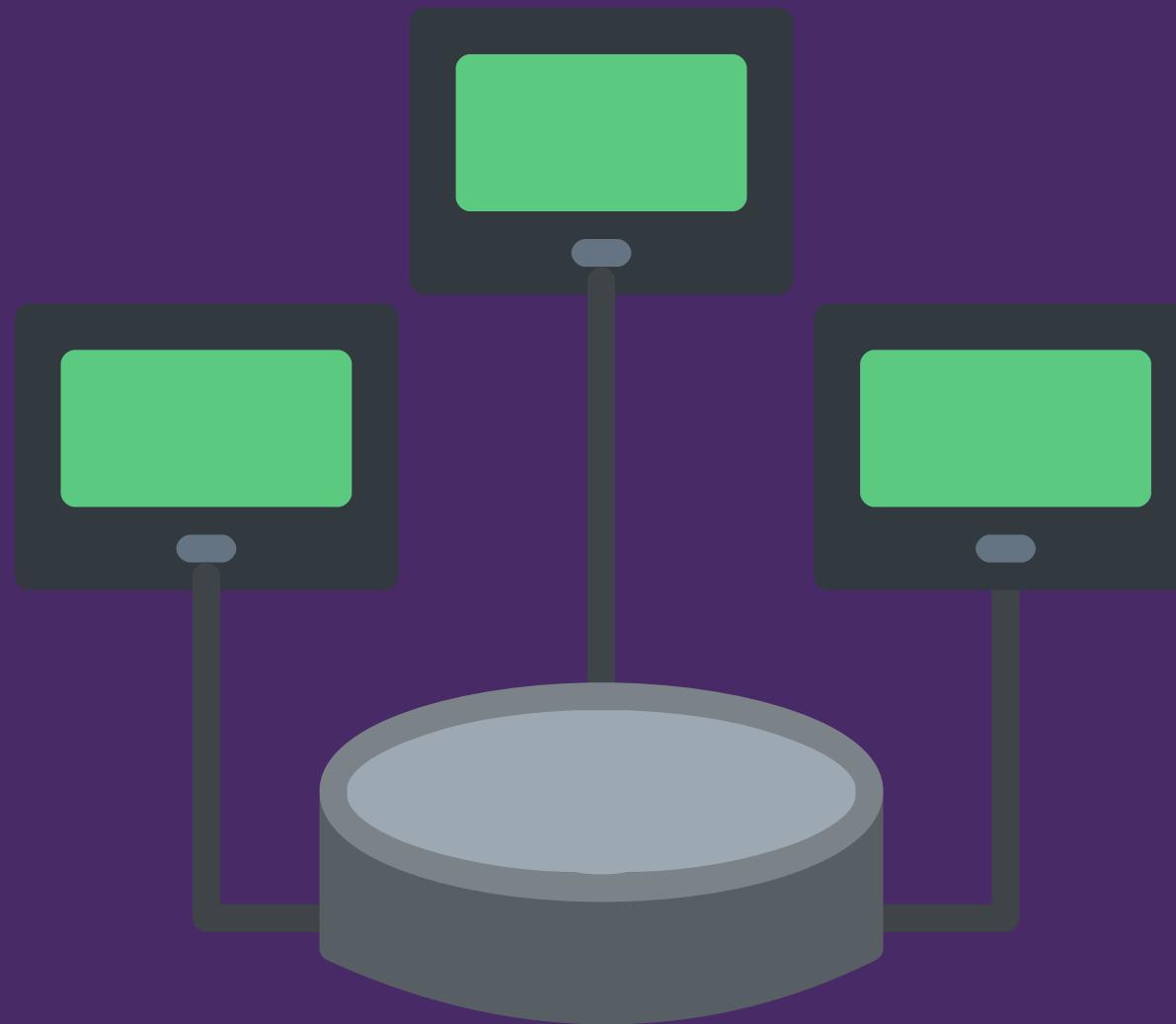




LABORATORIO SISTEMAS OPERATIVOS 2

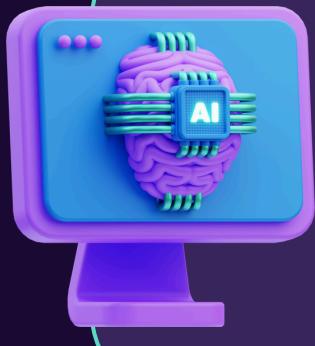
SISTEMAS DISTRIBUIDOS Y PROCEDIMIENTOS REMOTOS

SISTEMAS DISTRIBUIDOS



Un sistema distribuido es un conjunto de programas que aprovechan recursos computacionales en múltiples nodos para alcanzar un objetivo común. En estos sistemas, también conocidos como "computación distribuida" o "bases de datos distribuidas", los nodos se comunican y sincronizan a través de una red compartida. Los nodos pueden ser dispositivos de hardware físico o procesos de software independientes, e incluso otros sistemas encapsulados. La meta principal de los sistemas distribuidos es reducir cuellos de botella y eliminar puntos únicos de falla, aumentando así la eficiencia, resiliencia y escalabilidad del sistema en su conjunto.

EJEMPLOS COMUNES



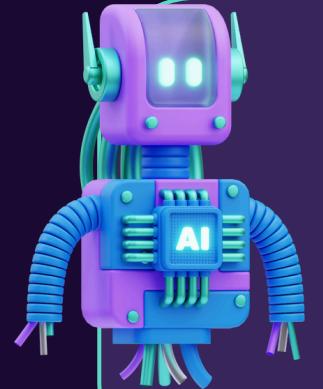
Internet

Una red global de computadoras conectadas que permite la transmisión de información a nivel mundial, donde cada nodo puede funcionar de manera independiente y al mismo tiempo como parte de una red colaborativa.



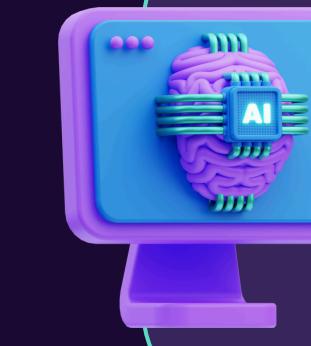
Servicios de mensajería

(p. ej., WhatsApp, Telegram): Las aplicaciones de mensajería dependen de sistemas distribuidos para enviar y recibir mensajes en tiempo real a través de redes de servidores distribuidos en distintas ubicaciones geográficas.



Aplicaciones en la nube

(p. ej., Google Drive, Dropbox): Servicios de almacenamiento y procesamiento donde los datos están distribuidos en varios servidores para aumentar la accesibilidad y redundancia.



Redes de distribución de contenido (CDN)

Sistemas como Akamai y Cloudflare distribuyen copias de contenido (videos, imágenes, archivos) en múltiples servidores alrededor del mundo, optimizando la disponibilidad y velocidad de acceso.

CARACTERÍSTICAS DE LOS SISTEMAS DISTRIBUIDOS

Cada una de estas propiedades es esencial para el diseño y funcionamiento de un sistema distribuido robusto

Transparencia

La capacidad de hacer que los aspectos de la distribución (ubicación, fallos, concurrencia) sean "invisibles" o no visibles para los usuarios.



01

TRANSPARENCIA DE ACCESO

Permite acceder a los recursos sin importar dónde estén ubicados.

02

TRANSPARENCIA DE UBICACIÓN

Los usuarios no necesitan saber la ubicación física de los recursos.

03

TRANSPARENCIA DE REPLICACIÓN

Múltiples copias de datos o servicios aparecen como una sola unidad para los usuarios.

04

TRANSPARENCIA DE CONCURRENCIA

Varios usuarios pueden interactuar con el sistema sin interferirse entre sí.

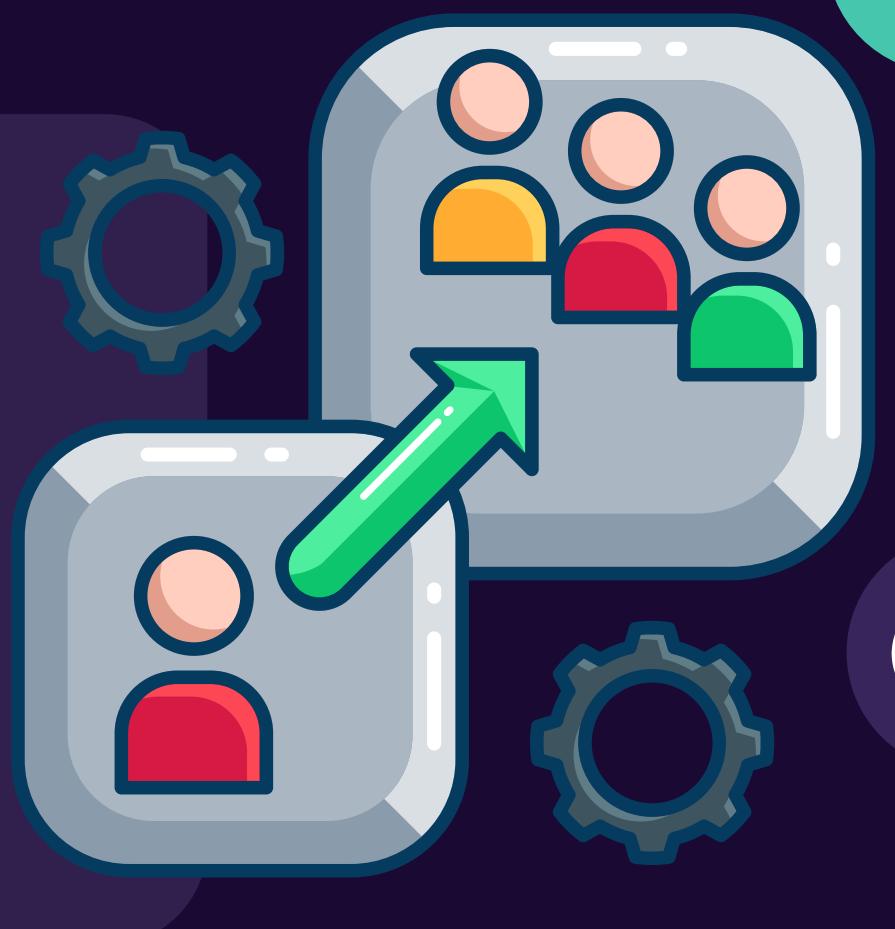
CARACTERÍSTICAS DE LOS SISTEMAS DISTRIBUIDOS

Cada una de estas propiedades es esencial para el diseño y funcionamiento de un sistema distribuido robusto

Escalabilidad

La capacidad del sistema para crecer y adaptarse a medida que la demanda de usuarios o recursos aumenta.

Redes sociales como Facebook y Twitter deben escalar sus sistemas para manejar miles de millones de usuarios activos y peticiones en tiempo real, sin comprometer el rendimiento



01

ESCALABILIDAD HORIZONTAL VS. VERTICAL

- **Horizontal:** Añadir más nodos al sistema (por ejemplo, agregar servidores adicionales para manejar más tráfico web).
- **Vertical:** Aumentar la capacidad de los nodos existentes (por ejemplo, añadir más memoria o procesadores a un servidor).

02

RETOS EN LA ESCALABILIDAD

- **Consistencia de datos:** Mantener la coherencia de la información entre nodos puede ser más complejo al agregar más nodos.
- **Latencia de comunicación:** A medida que el sistema crece, la distancia entre nodos también puede crecer, afectando el tiempo de respuesta.

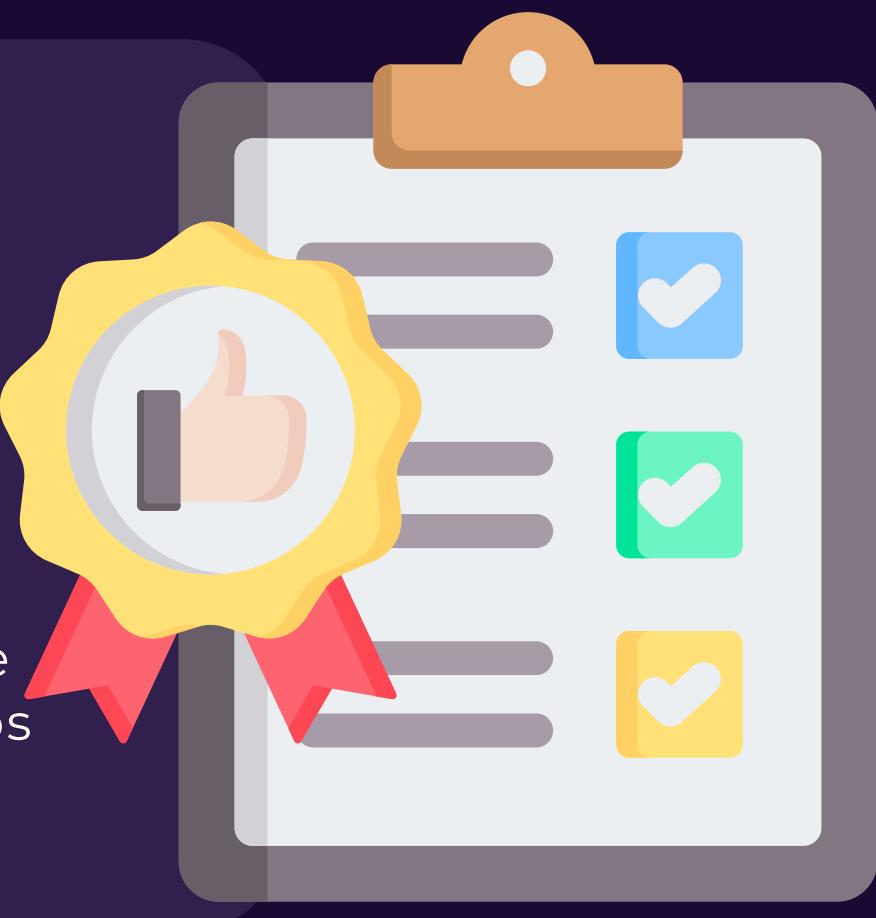
CARACTERÍSTICAS DE LOS SISTEMAS DISTRIBUIDOS

Cada una de estas propiedades es esencial para el diseño y funcionamiento de un sistema distribuido robusto

Confiabilidad

Capacidad del sistema para seguir funcionando correctamente incluso cuando ocurren fallos en ciertos componentes.

Netflix utiliza múltiples servidores en diversas ubicaciones para asegurarse de que el contenido esté disponible para los usuarios incluso si algunos servidores fallan.



01

TOLERANCIA A FALLOS

Los sistemas distribuidos a menudo incluyen mecanismos de redundancia y recuperación que permiten continuar con las operaciones a pesar de errores o caídas de ciertos nodos.

02

REDUNDANCIA Y RECUPERACIÓN

- **Redundancia:** Duplicación de datos y servicios en varios nodos.
- **Recuperación ante fallos:** Estrategias para reanudar el servicio lo antes posible después de un fallo (p. ej., balanceo de carga, copias de seguridad automáticas).

CARACTERÍSTICAS DE LOS SISTEMAS DISTRIBUIDOS

Cada una de estas propiedades es esencial para el diseño y funcionamiento de un sistema distribuido robusto

Concurrencia

La capacidad de manejar múltiples procesos o peticiones simultáneas en el sistema, facilitando el uso compartido de recursos por varios usuarios al mismo tiempo.

En bases de datos distribuidas como las utilizadas por bancos, la concurrencia asegura que varias transacciones puedan realizarse simultáneamente sin que los datos pierdan consistencia o integridad.



SINCRONIZACIÓN Y CONSISTENCIA

- **Sincronización:** Controla la interacción de múltiples procesos que pueden compartir recursos o información.
- **Consistencia:** Asegura que todos los usuarios vean una misma versión de los datos en un entorno distribuido.

CONTROL DE CONCURRENCIA

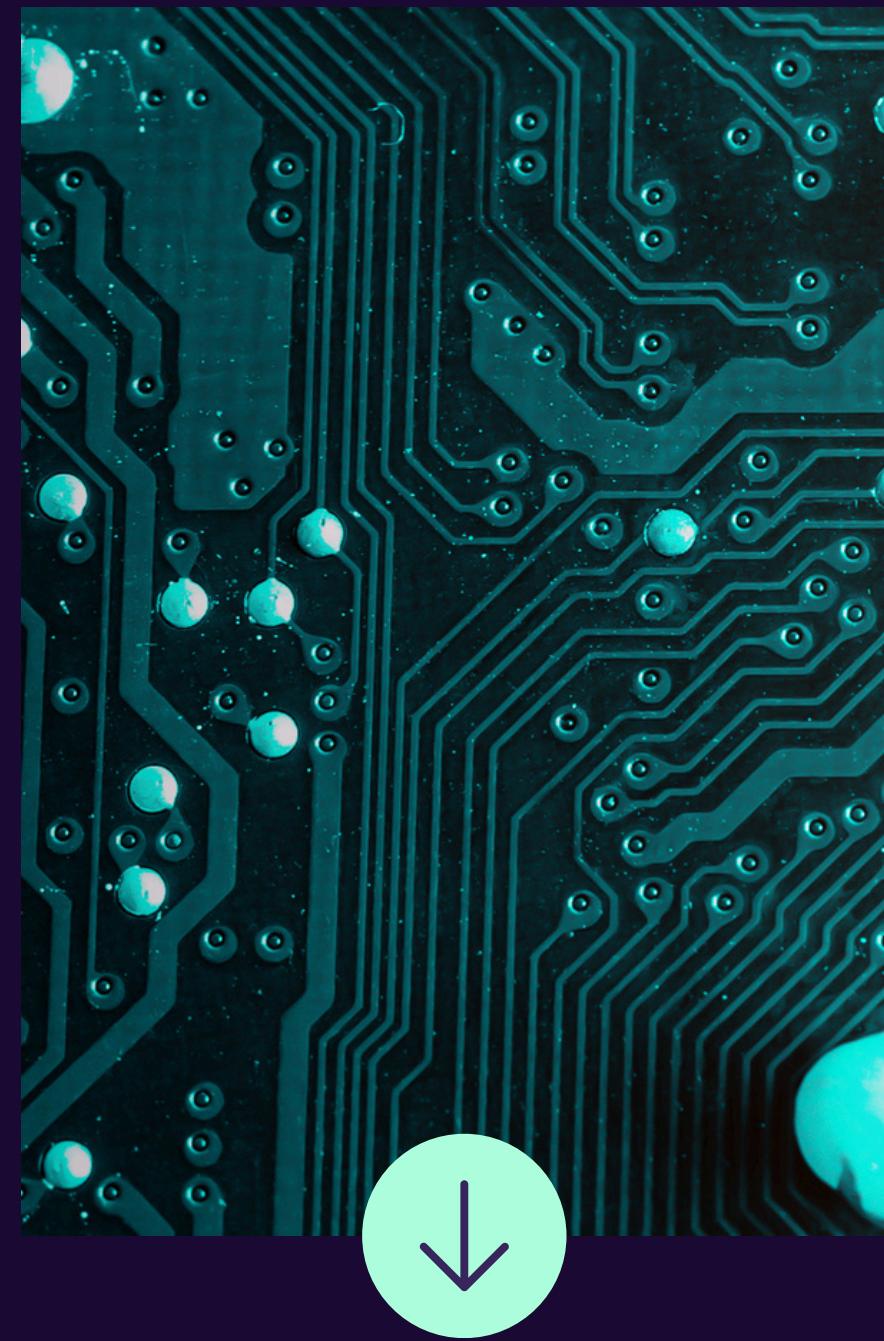
- Evita condiciones de carrera y garantiza la integridad de los datos.
- Puede implicar mecanismos como bloqueos, semáforos, y algoritmos de consenso en sistemas distribuidos.

LLAMADAS A PROCEDIMIENTOS REMOTOS (RPC)

Las Llamadas a Procedimientos Remotos (RPC, por sus siglas en inglés) son un mecanismo que permite a un programa ejecutar una función o procedimiento en una computadora remota como si fuera una llamada local. Este enfoque facilita la comunicación y la colaboración entre diferentes sistemas dentro de un entorno distribuido.

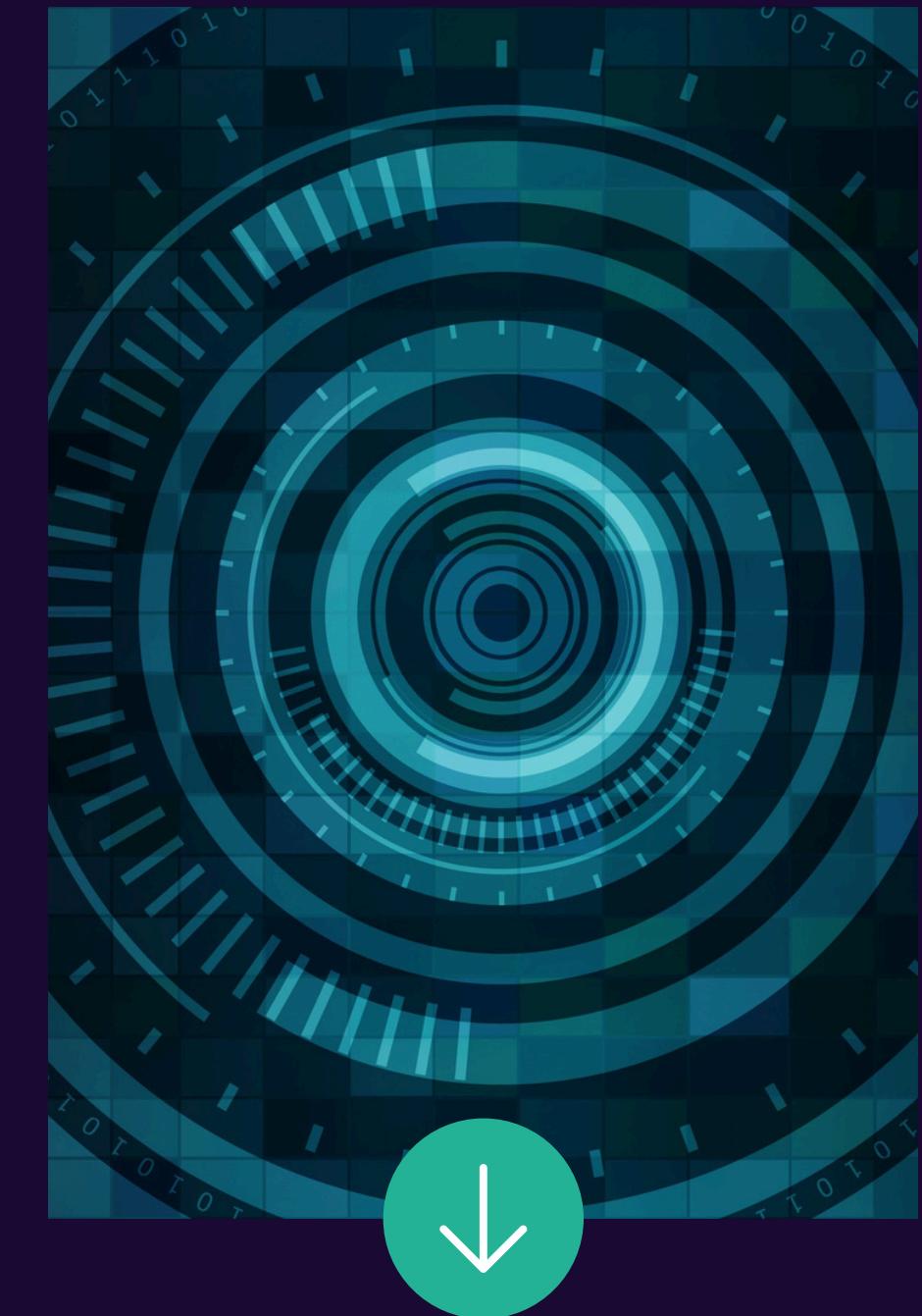
OBJETIVO DE LAS RPC

- Simplificar la interacción entre sistemas distribuidos.
- Ocultar la complejidad de la comunicación de red al programador.



¿CÓMO FUNCIONA?

- El cliente invoca una función local que, en realidad, envía una solicitud al servidor.
- El servidor ejecuta la función solicitada y devuelve el resultado al cliente.



IMPORTANCIA EN LOS SISTEMAS DISTRIBUIDOS

- Permiten la construcción de aplicaciones distribuidas modulares y escalables.
- Facilitan la reutilización de servicios y recursos en diferentes partes de una red.

FUNCIONAMIENTO BÁSICO DE RPC



01 CLIENTE HACE UNA SOLICITUD

- El cliente invoca una función local que actúa como un proxy.
- **Stub del Cliente:** Una interfaz que prepara la solicitud para enviarla al servidor.



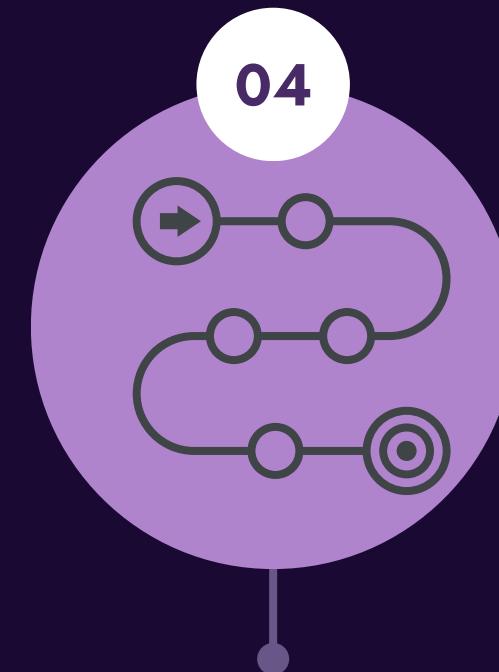
02 MARSHALLING

Serialización de los parámetros de la función para su transmisión a través de la red.



03 TRANSMISIÓN DE LA SOLICITUD

Los datos serializados se envían al servidor a través de la red.

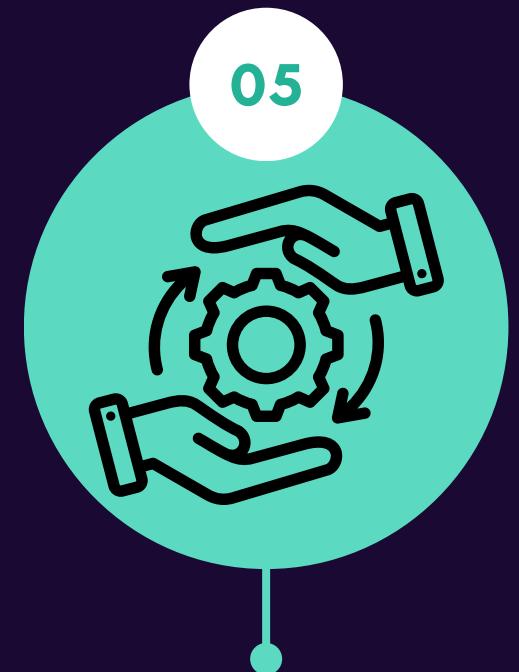


04 SERVIDOR PROCESA LA SOLICITUD

Stub del Servidor: Recibe la solicitud, deserializa los parámetros y ejecuta la función solicitada.

Cliente Stub: Representa la interfaz del servidor en el cliente.
Servidor Stub: Representa la interfaz del cliente en el servidor.

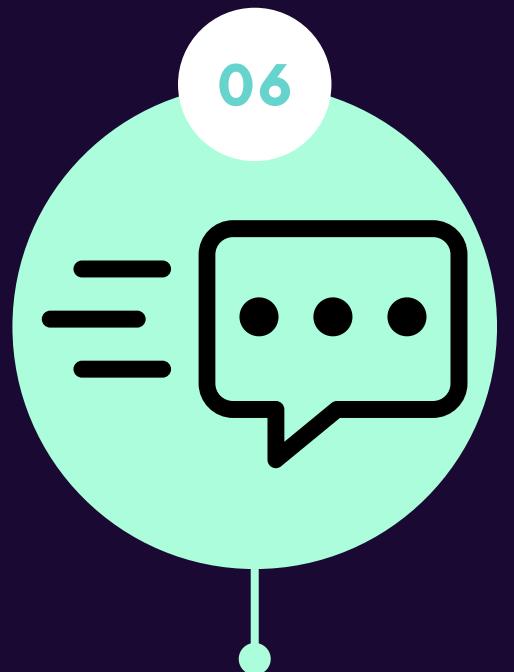
FUNCIONAMIENTO BÁSICO DE RPC



05

UNMARSHALLING

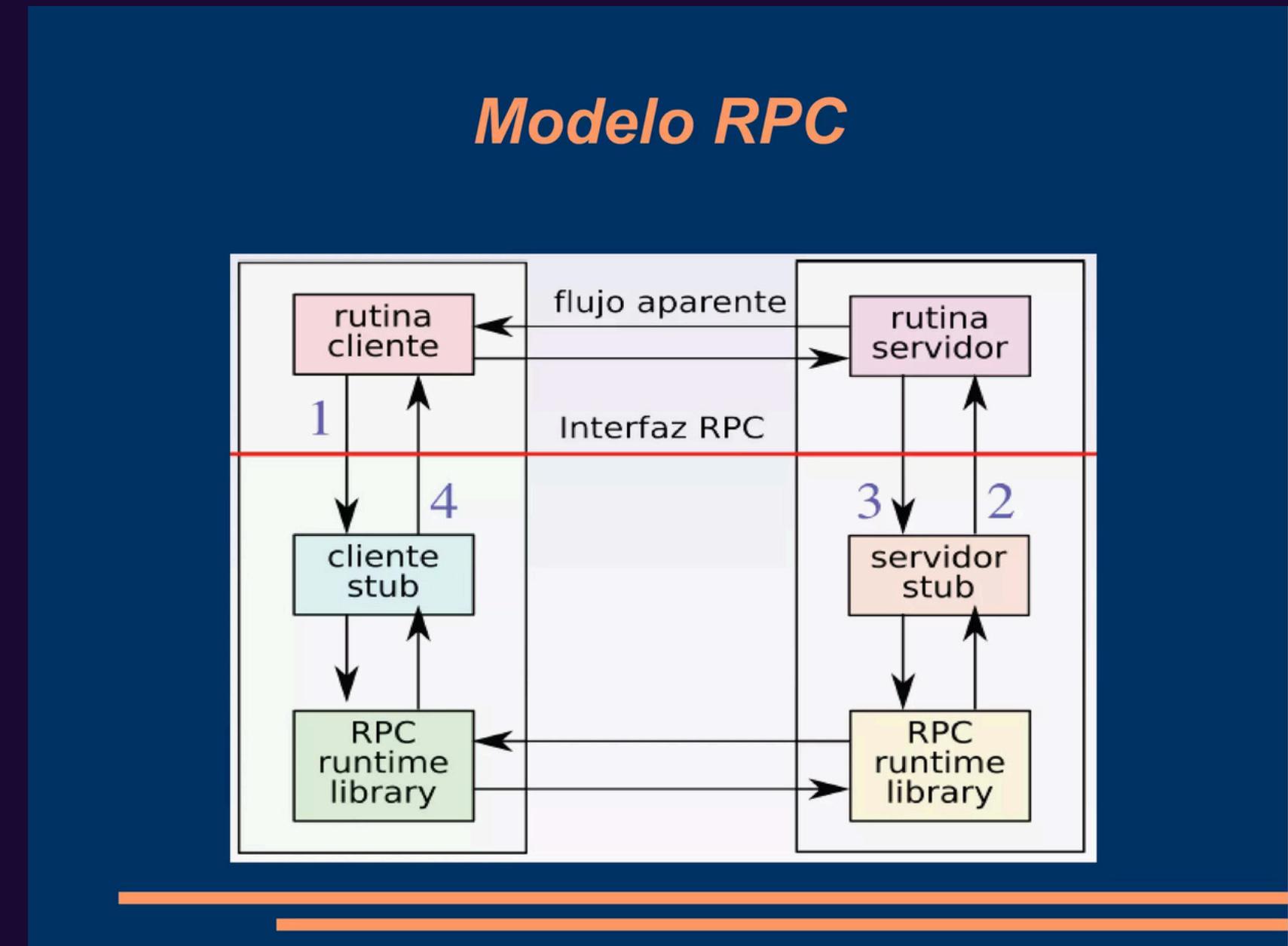
Deserialización del resultado para enviarlo de vuelta al cliente.



06

RESPUESTA AL CLIENTE

El cliente recibe el resultado y lo procesa como si fuera una llamada local.



- **Marshalling:** Proceso de convertir los parámetros de la función en un formato que pueda ser transmitido a través de la red.
- **Unmarshalling:** Proceso de convertir los datos recibidos de vuelta a su formato original para su uso en el programa.

PROTOCOLOS DE RPC



gRPC

Un framework de RPC de alto rendimiento desarrollado por Google, basado en HTTP/2.

- Soporte para múltiples lenguajes, streaming bidireccional, eficiente en el uso de recursos.
- **Casos de Uso:** Microservicios, comunicación entre servicios en la nube.



SOAP (Simple Object Access Protocol):

Un protocolo basado en XML para intercambiar información estructurada en la implementación de servicios web.

- Extensible, independiente del protocolo de transporte, alto nivel de seguridad.
- **Casos de Uso:** Servicios empresariales, aplicaciones que requieren transacciones seguras y fiables.



XML-RPC

Un protocolo que utiliza XML para codificar sus llamadas y HTTP como protocolo de transporte.

- Simplicidad, fácil de implementar.
- **Casos de Uso:** Integración de sistemas, aplicaciones web simples.



JSON-RPC

Similar a XML-RPC pero utiliza JSON para la codificación.

- Ligero, fácil de leer y escribir, compatible con aplicaciones web modernas.
- **Casos de Uso:** Aplicaciones web y móviles, APIs RESTful.

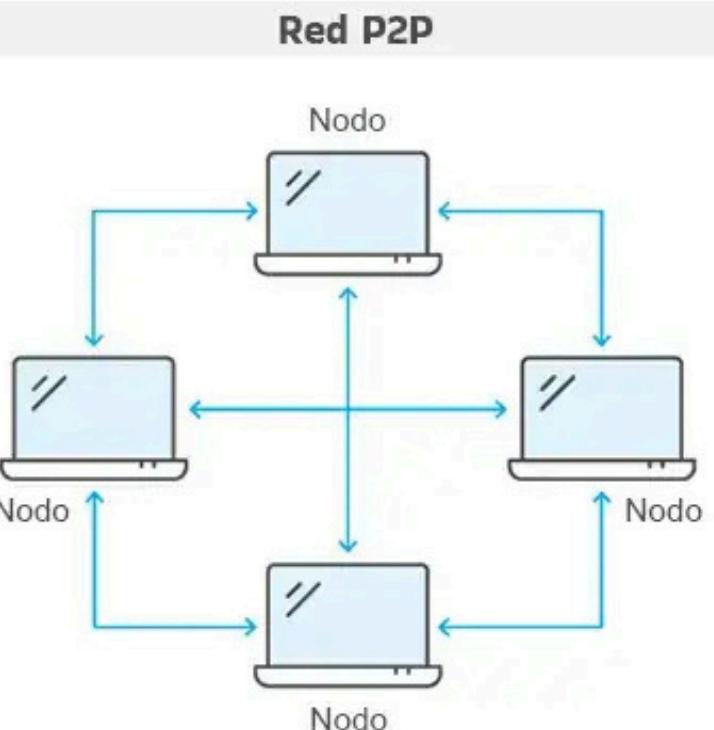
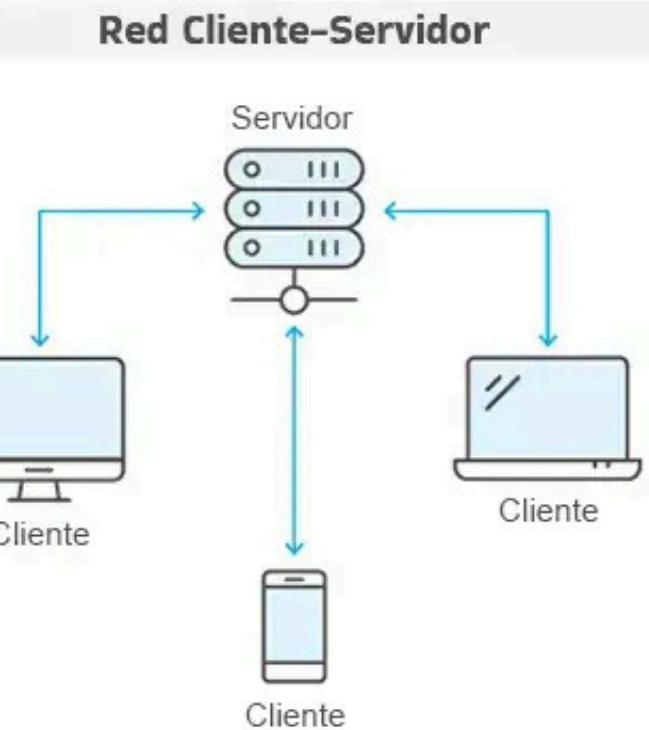


¿CUÁL USAR?

- **gRPC:** Comunicación entre microservicios en una arquitectura de Kubernetes.
- **SOAP:** Servicios web financieros que requieren alta seguridad y transacciones confiables.
- **JSON-RPC:** APIs para aplicaciones móviles que requieren respuestas rápidas y ligeras.

MODELOS DE ARQUITECTURA EN SISTEMAS DISTRIBUIDOS

Cliente-Servidor vs P2P



01

CLIENTE SERVIDOR

La red cliente-servidor está compuesta por al menos un servidor central que controla la red y una serie de dispositivos cliente que se conectan al servidor para realizar tareas específicas.

Ejemplos típicos: servidores web y bases de datos.

02

PEER-TO-PEER (P2P)

Descripción del modelo en el que cada nodo actúa como cliente y servidor, permitiendo que comparten recursos de manera descentralizada.

Ejemplos comunes: redes de intercambio de archivos y blockchain.

VIDEO

RPC en un sistema distribuido



CONSENSO Y COORDINACIÓN EN SISTEMAS DISTRIBUIDOS

¿QUÉ ES EL CONSENSO?

El consenso en sistemas distribuidos se refiere a la capacidad de **varios nodos** (o máquinas) para llegar a un acuerdo sobre un valor o estado compartido, a pesar de fallas o discrepancias.

NECESIDAD

Es fundamental para mantener la consistencia en sistemas distribuidos, donde la comunicación **no es siempre fiable**.

PROBLEMAS DE CONSENSO

- **Elecciones en una Red:** Elegir un nodo líder en un grupo distribuido, por ejemplo, para coordinar acciones en un clúster.
- **Consistencia en Bases de Datos Distribuidas:** Asegurar que todas las réplicas de datos estén sincronizadas y representen el mismo estado, aun cuando se produzcan fallos.

IMPORTANCIA

El consenso asegura que el sistema pueda continuar operando correctamente y ofrece una única fuente de "**verdad**" en un entorno de múltiples nodos.

COORDINACIÓN EN SISTEMAS DISTRIBUIDOS



Importancia de la Coordinación y Sincronización

- Los sistemas distribuidos requieren que múltiples nodos trabajen juntos de forma coordinada para evitar problemas como la doble escritura o los conflictos de datos.
- La coordinación permite que los procesos y nodos realicen tareas sincronizadamente, gestionando el acceso a los recursos y manteniendo la consistencia de datos en tiempo real.



Coordinación en Sistemas de Archivos Distribuidos

Google File System (GFS): GFS necesita coordinación para que los nodos puedan leer y escribir archivos de manera sincronizada, evitando la corrupción de datos y manteniendo la integridad del sistema.

GFS utiliza un nodo central de control (master) que coordina qué fragmentos del archivo deben escribirse y en qué orden. Otros nodos de almacenamiento actúan como réplicas para garantizar la disponibilidad y consistencia de los datos.



**¡GRACIAS POR
LA ATENCIÓN!**

¿Dudas?