EcomLabs

GUÍA PASO A PASO

Server y Client Components en Next.js: el árbol

El diagrama del árbol de componentes, la tabla para decidir si un componente va en el servidor o en el cliente y un refactor completo de una ficha de producto: de una página de cliente con useEffect a un Server Component con hojas interactivas. Con la checklist.

PARA	Proyectos en Next.js con App Router (React 19)
PROBADO CON	Next.js 15.5.23 · React 19.0.0 · Node 22.22.3 · next build, next start, curl y Playwright en local
ACTUALIZADA	19 de noviembre de 2026
DOCUMENTACIÓN	nextjs.org/docs/app/getting-started/server-and-client-components
ARTÍCULO	ecomlabs.dev/g/rsc
AUTOR	Luis Morales · @luismoralesdev

Índice

Paso 01	El diagrama del árbol
Paso 02	La tabla: ¿servidor o cliente?
Paso 03	Antes: la página entera en el cliente
Paso 04	Después: servidor con hojas de cliente
Paso 05	Compruébalo en tu proyecto
Al terminar	Checklist

PASO 01

El diagrama del árbol

En el App Router todo es Server Component por defecto (no hay directiva; `'use server'` es para Server Functions). El código cruza al cliente por imports y los datos por props serializables. Este es el árbol de la ficha de la demo; dibuja el tuyo con la misma forma: raíz y ramas en el servidor, cliente solo en las hojas.

```
layout.tsx                  servidor
+-- page.tsx (async)        servidor, dueño de Reviews
    +-- h1, precio          servidor
    +-- AddToCart           CLIENTE (hoja)
    +-- Tabs                CLIENTE (hoja)
        +-- Reviews (async) servidor, llega como prop
```

Dueño y padre. `page.tsx` escribe `<Reviews />` en su JSX: es su dueño. `Tabs` lo contiene en pantalla: es su padre. Como el dueño es de servidor, `Reviews` se renderiza en el servidor y `Tabs` recibe solo su salida.

Qué viaja al navegador en la primera carga: el **HTML**, con la página ya pintada y el texto de las reseñas; el **RSC Payload**, con la salida de los Server Components, referencias a los Client Components y sus props; y el **JavaScript** de `AddToCart` y `Tabs` (965 bytes sin comprimir en la demo). No viajan el código de `Reviews`, la función que lee los datos ni los campos que no pasaste como prop.

PASO 02

La tabla: ¿servidor o cliente?

SI EL COMPONENTE…	VA EN
Lee la base de datos, archivos o una clave secreta	Servidor
Usa una librería pesada solo para pintar (Markdown, fechas)	Servidor
Solo muestra datos: título, precio, descripción, reseñas	Servidor
Abre y cierra con `<details>` o envía un `<form>` a una Server Function	Servidor
Usa `useState`, `onClick` u `onChange`	Cliente
Usa `useEffect` (revisa antes si de verdad hace falta)	Cliente
Lee `window`, `localStorage` o `navigator`	Cliente
Usa un hook propio que hace lo anterior	Cliente
Da un contexto (provider)	Cliente, envolviendo solo `{children}`

Si no cumple ninguna fila de cliente, déjalo en el servidor. Si cumple una, saca solo esa parte a su propio archivo con `"use client"`.

PASO 03

Antes: la página entera en el cliente

Una ficha que pide el producto a una ruta de API al montar. Compila y funciona, pero todo es cliente:

```
"use client";

import { useEffect, useState } from "react";
import { useParams } from "next/navigation";

type Product = { id: string; name: string; price: string; sizes: string[] };

export default function Page() {
  const { id } = useParams<{ id: string }>();
  const [product, setProduct] = useState<Product | null>(null);
  const [size, setSize] = useState("");

  useEffect(() => {
    fetch(`/api/products/${id}`)
      .then((r) => r.json())
      .then((p: Product) => {
        setProduct(p);
        setSize(p.sizes[0]);
      });
  }, [id]);

  if (!product) return <p>Cargando…</p>;

  return (
    <main>
      <h1>{product.name}</h1>
      <p>{product.price}</p>
      {product.sizes.map((s) => (
        <button key={s} aria-pressed={s === size} onClick={() => setSize(s)}>
          {s}
        </button>
      ))}
    </main>
  );
}
```

Qué medimos en la demo con `next start` y `curl`:

- El HTML inicial solo trae `<p>Cargando…</p>`. El nombre del producto aparece después de hidratar y hacer la petición; una imagen principal llegaría igual de tarde y empeoraría el LCP.
- Hace falta una ruta de API pública solo para alimentar esta página.
- Todo el componente es JavaScript del navegador, aunque lo único interactivo son los botones de talla.

PASO 04

Después: servidor con hojas de cliente

La página lee el producto en el servidor y pasa a cada hoja solo lo que usa. `lib/products.ts` empieza con `import "server-only"`.

app/p/[id]/page.tsx

```
import { notFound } from "next/navigation";
import { getProduct } from "@/lib/products";
import AddToCart from "./add-to-cart";
import Tabs from "./tabs";
import Reviews from "./reviews";

export default async function Page({ params }: { params: Promise<{ id: string }> }) {
  const { id } = await params;
  const product = await getProduct(id);
  if (!product) notFound();

  const price = new Intl.NumberFormat("es-MX", { style: "currency", currency: "MXN" })
    .format(product.priceCents / 100);

  return (
    <main>
      <h1>{product.name}</h1>
      <p>{price}</p>
      <AddToCart productId={product.id} sizes={product.sizes} />
      <Tabs
        panels={[
          { label: "Reseñas", content: <Reviews productId={product.id} /> },
          { label: "Envío", content: <p>Envío gratis desde $999.</p> },
        ]}
      />
    </main>
  );
}
```

`tabs.tsx` es un componente de cliente que recibe los paneles como prop y nunca importa `Reviews`: el código completo está en el artículo. `add-to-cart.tsx` lleva `"use client"` y los botones de talla del paso anterior, ahora con las tallas como prop. `reviews.tsx` es un Server Component `async` que lee las reseñas y devuelve una lista. Resultado: el HTML trae el nombre, el precio y las reseñas; la ruta de API ya no hace falta; el JavaScript de la página es solo `AddToCart` y `Tabs`.

> **CUIDADO**
> No pases el producto completo a una hoja. En la demo, `<AddToCart product={product} />` publicó un campo interno (`costCents`) en el payload y en el HTML. TypeScript no lo detecta. Tampoco pases funciones normales como props: el build falla al prerenderizar.

PASO 05

Compruébalo en tu proyecto

Con `npm run build` y `npm start` corriendo, cambia la ruta y el campo por los tuyos:

```
curl -s http://localhost:3000/p/1 | grep -c costCents
curl -s -H "RSC: 1" http://localhost:3000/p/1 | grep -c costCents
grep -rl "getReviews" .next/static
```

Las dos primeras líneas cuentan cuántas veces aparece el campo interno en el HTML y en el payload: debe ser `0`. La tercera busca el nombre de una función de servidor en los archivos públicos del build: no debe listar nada. El formato del payload es interno de React; úsalo solo para revisar, no para depender de él.

AL TERMINAR

Checklist

- [ ] Páginas y layouts sin `'use client'`.
- [ ] Cada componente de cliente cumple una fila de cliente de la tabla.
- [ ] Lo interactivo vive en hojas pequeñas, cada una en su archivo.
- [ ] El contenido de servidor entra al cliente como `children` u otra prop, nunca como import.
- [ ] Las hojas reciben solo los campos que usan.
- [ ] Ninguna función normal se pasa como prop a un componente de cliente.
- [ ] Los archivos que leen datos o claves empiezan con `import "server-only"`.
- [ ] Los datos de la primera carga los lee un Server Component, no un `useEffect`.
- [ ] El HTML y el payload de una página no contienen campos internos.