EcomLabs

GUÍA PASO A PASO

LCP: la imagen principal primero

Cómo encontrar el elemento del LCP de tu tienda, quitarle la carga diferida, subirle la prioridad y servirle el tamaño correcto, en HTML y en Liquid de Shopify. Con un checklist y los fragmentos de código listos para copiar.

PARA	Cualquier tienda en HTML o un tema de Shopify
PROBADO CON	Lighthouse 13.5.0 · Chrome 156 · Shopify CLI 3.88.1 (theme check)
ACTUALIZADA	30 de octubre de 2026
DOCUMENTACIÓN	web.dev/articles/optimize-lcp
ARTÍCULO	ecomlabs.dev/g/lcp
AUTOR	Luis Morales · @luismoralesdev

Índice

Antes de empezar	Qué vas a arreglar
Paso 01	Encuentra el elemento del LCP
Paso 02	Quita la carga diferida
Paso 03	Sube la prioridad con fetchpriority
Paso 04	Sirve el tamaño y el formato correctos
Paso 05	Lo mismo en un tema de Shopify
Paso 06	Mide antes y después
Al terminar	Checklist

ANTES DE EMPEZAR

Qué vas a arreglar

El LCP (Largest Contentful Paint) es el momento en que se pinta el elemento más grande de la pantalla. En una tienda suele ser la imagen principal de la portada o la foto del producto. Google lo considera bueno hasta 2,5 segundos, medido en el percentil 75 de las visitas reales, con móvil y escritorio por separado.

LCP	VEREDICTO	QUÉ HACER
2,5 s o menos	Bueno	Mantenerlo: revisa cada cambio de tema o de app
Entre 2,5 y 4 s	Necesita mejora	Aplica los pasos 02 a 04
Más de 4 s	Malo	Empieza por el peso de la imagen (paso 04)

La guía sigue el orden del desglose del LCP: primero que la imagen se pida pronto (retraso de carga) y después que baje rápido (duración de carga).

PASO 01

Encuentra el elemento del LCP

Abre tu página en Chrome, activa la emulación de celular en DevTools, pega esto en la consola y recarga:

```
new PerformanceObserver((list) => {
  const entries = list.getEntries();
  const last = entries[entries.length - 1];
  console.log("LCP", Math.round(last.startTime), "ms", last.element);
}).observe({ type: "largest-contentful-paint", buffered: true });
```

La última línea que imprima es el LCP hasta ese momento. Pasa el cursor sobre el elemento en la consola para verlo marcado en la página. Si es un texto y no una imagen, esta guía no aplica a ese elemento: revisa el servidor y lo que bloquea el renderizado.

Para el desglose, graba una carga en el panel **Performance** y abre **Insights**. En el panel **Network**, activa la columna **Priority** para ver con qué prioridad salió la imagen.

PASO 02

Quita la carga diferida

Busca en el HTML que llega al navegador (Ver código fuente, no el código del tema) la etiqueta de tu imagen principal. Esto es lo que no debe tener:

```
<img src="/img/hero-2400.jpg" loading="lazy" alt="Taza de cerámica azul sobre una mesa de madera">
```

- Quita `loading="lazy"`, o cámbialo por `loading="eager"`, que es lo mismo que no ponerlo.

- Si ves `data-src` o una clase como `lazyload`, hay una librería de JavaScript ocultando la URL. Pasa la URL a `src` en la imagen principal.
- Deja `loading="lazy"` en las imágenes de más abajo: ahí sí ayuda.

> **CUIDADO**
>
> En Shopify, `image_tag` pone `loading="lazy"` solo en secciones más abajo en la página. Si tu banner no es la primera sección, o una app de velocidad añade lazy a todas las imágenes, revisa el HTML final.

PASO 03

Sube la prioridad con fetchpriority

Las imágenes empiezan con prioridad baja y el navegador solo sube las que ve en pantalla cuando termina el diseño. `fetchpriority="high"` se salta esa espera. Reglas:

- Una sola imagen por página: la del LCP.
- Nunca junto a `loading="lazy"`.
- Es una pista para el navegador, no una orden. Comprueba en la columna Priority que salió en `High`.
- Si la imagen es un fondo en CSS, el navegador no la ve al leer el HTML: pásala a `<img>` o precárgala con `<link rel="preload" as="image">` y `fetchpriority="high"`.

PASO 04

Sirve el tamaño y el formato correctos

Exporta la imagen en varios anchos (por ejemplo 480, 800 y 1200 px) en WebP o AVIF y deja que el navegador elija. Esta es la etiqueta completa de nuestra demo, con los tres arreglos:

```
<img
  src="/img/hero-800.webp"
  srcset="/img/hero-480.webp 480w, /img/hero-800.webp 800w, /img/hero-1200.webp 1200w"
  sizes="100vw"
  width="1200"
  height="675"
  fetchpriority="high"
  alt="Taza de cerámica azul sobre una mesa de madera">
```

ATRIBUTO	PARA QUÉ
`srcset`	Las versiones disponibles con su ancho real en píxeles.
`sizes`	Cuánto ocupa la imagen en pantalla. Sin él, el navegador asume `100vw`.
`width` y `height`	La proporción: reservan el espacio y evitan saltos (CLS).
`fetchpriority`	La prioridad del paso 03.

PASO 05

Lo mismo en un tema de Shopify

En Shopify no generas los archivos: `image_url` redimensiona en la CDN y Shopify elige el formato según el navegador. Para una sección de banner, decide la carga con `section.index`:

```
{%- liquid
  assign hero_loading = 'eager'
  assign hero_priority = 'auto'
  if section.index == 1
    assign hero_priority = 'high'
  elsif section.index > 3
    assign hero_loading = 'lazy'
  endif
-%}
<div class="hero-banner">
  {%- if section.settings.image != blank -%}
    {{ section.settings.image
      | image_url: width: 1600
      | image_tag:
          loading: hero_loading,
          fetchpriority: hero_priority,
          widths: '600, 900, 1200, 1600',
          sizes: '100vw'
    }}
  {%- endif -%}
</div>
```

Para la foto principal de la página de producto, cuando está en la primera sección:

```
<div class="product-media">
  {{ product.featured_image
    | image_url: width: 1200
    | image_tag:
        loading: 'eager',
        fetchpriority: 'high',
        widths: '400, 600, 800, 1000, 1200',
        sizes: '(min-width: 990px) 50vw, 100vw'
  }}
</div>
```

> **CUIDADO**
>
> `section.index` es `nil` en el editor de temas, en secciones estáticas y en la Section Rendering API. No escribas `section.index <= 3` para decidir lo eager: en esos casos la comparación falla y la imagen acaba en lazy. Pregunta por `section.index > 3` para el lazy y deja que el resto caiga en eager.

Los dos fragmentos pasaron `shopify theme check` en un tema mínimo. Pruébalos en un tema de desarrollo antes de publicarlos y ajusta `sizes` a tu diseño.

PASO 06

Mide antes y después

Corre Lighthouse antes de tocar nada y después de cada paso, siempre con la misma configuración. Así medimos nuestra demo (Lighthouse instalado con `npm install -g lighthouse`):

```
lighthouse http://127.0.0.1:4173/0-antes.html --only-categories=performance --throttling-method=devtools --chrome-flags="--headless=new" --output=json --output-path=antes.json
```

Cambia la URL por la de tu página. `--throttling-method=devtools` limita de verdad la red y la CPU durante la carga; con la simulación por defecto, en nuestra demo el lazy no se notó. Anota la mediana de tres corridas:

MOMENTO	LCP CORRIDA 1	CORRIDA 2	CORRIDA 3	MEDIANA
Antes				
Sin lazy				
Con fetchpriority				
Tamaño y formato				

Es laboratorio: sirve para comparar cambios. El veredicto real llega con los datos de campo, en el informe de Core Web Vitals de Search Console, unas semanas después.

AL TERMINAR

Checklist

- [ ] Sabes cuál es el elemento del LCP en la portada y en la página de producto.
- [ ] La imagen principal no tiene `loading="lazy"` ni `data-src`.
- [ ] Tiene `fetchpriority="high"` y ninguna otra imagen lo tiene.
- [ ] Tiene `srcset`, un `sizes` real, `width` y `height`.
- [ ] El celular baja una versión de su ancho, en WebP o AVIF.
- [ ] En Shopify, el lazy depende de `section.index > 3`.
- [ ] Mediste antes y después con la misma configuración.
- [ ] Agendaste revisar Core Web Vitals en Search Console.