EcomLabs

STEP-BY-STEP GUIDE

LCP: hero image first

How to find your store's LCP element, stop lazy-loading it, raise its priority and serve it at the right size, in plain HTML and in Shopify Liquid. With a checklist and code snippets ready to copy.

FOR	Any HTML store or a Shopify theme
TESTED WITH	Lighthouse 13.5.0 · Chrome 156 · Shopify CLI 3.88.1 (theme check)
UPDATED	October 30, 2026
DOCS	web.dev/articles/optimize-lcp
ARTICLE	ecomlabs.dev/g/lcp-en
AUTHOR	Luis Morales · @luismoralesdev

Contents

Before you start	What you are fixing
Step 01	Find your LCP element
Step 02	Stop lazy-loading it
Step 03	Raise the priority with fetchpriority
Step 04	Serve the right size and format
Step 05	The same in a Shopify theme
Step 06	Measure before and after
Final	Checklist

BEFORE YOU START

What you are fixing

LCP (Largest Contentful Paint) is the moment the largest element on screen gets painted. On a store it is usually the home page hero or the product photo. Google counts 2.5 seconds or less as good, measured at the 75th percentile of real visits, with mobile and desktop counted separately.

LCP	RATING	WHAT TO DO
2.5 s or less	Good	Keep it: recheck after every theme or app change
Between 2.5 and 4 s	Needs improvement	Work through steps 02 to 04
Over 4 s	Poor	Start with the image's weight (step 04)

The guide follows the LCP breakdown: first get the image requested early (load delay), then get it downloaded fast (load duration).

STEP 01

Find your LCP element

Open your page in Chrome, turn on mobile emulation in DevTools, paste this into the console and reload:

```
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 });
```

The last line it logs is the LCP so far. Hover the element in the console to highlight it on the page. If it is text rather than an image, this guide does not apply to that element: look at the server and at what blocks rendering.

For the breakdown, record a load in the **Performance** panel and open **Insights**. In the **Network** panel, turn on the **Priority** column to see which priority the image got.

STEP 02

Stop lazy-loading it

Find your hero's tag in the HTML the browser receives (View source, not the theme code). This is what it must not have:

```
<img src="/img/hero-2400.jpg" loading="lazy" alt="Blue ceramic mug on a wooden table">
```

- Remove `loading="lazy"`, or change it to `loading="eager"`, which is the same as leaving it out.

- If you see `data-src` or a class like `lazyload`, a JavaScript library is hiding the URL. Move the URL to `src` on the hero.
- Keep `loading="lazy"` on images further down: there it helps.

> **CAUTION**
> On Shopify, `image_tag` sets `loading="lazy"` only on sections further down the page. If your banner is not the first section, or a speed app lazy-loads every image, check the final HTML.

STEP 03

Raise the priority with fetchpriority

Images start at low priority and the browser only bumps the ones on screen once layout is done. `fetchpriority="high"` skips that wait. Rules:

- One image per page: the LCP one.
- Never together with `loading="lazy"`.
- It is a hint to the browser, not a directive. Check in the Priority column that it went out as `High`.
- If the image is a CSS background, the browser does not see it while parsing the HTML: move it to an `<img>` or preload it with `<link rel="preload" as="image">` and `fetchpriority="high"`.

STEP 04

Serve the right size and format

Export the image at several widths (for example 480, 800 and 1200 px) as WebP or AVIF and let the browser choose. This is the full tag from our demo, with all three fixes:

```
<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="Blue ceramic mug on a wooden table">
```

ATTRIBUTE	WHAT FOR
`srcset`	The available files with their real pixel widths.
`sizes`	How wide the image renders. Without it, the browser assumes `100vw`.
`width` and `height`	The aspect ratio: they reserve the space and prevent layout jumps (CLS).
`fetchpriority`	The priority from step 03.

STEP 05

The same in a Shopify theme

On Shopify you do not make the files: `image_url` resizes on the CDN and Shopify picks the format per browser. For a banner section, decide loading with `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>
```

For the main photo on a product page, when it sits in the first section:

```
<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>
```

> **CAUTION**
>
> `section.index` is `nil` in the theme editor, in static sections and in the Section Rendering API. Do not write `section.index <= 3` to pick eager: in those cases the comparison fails and the image ends up lazy. Test `section.index > 3` for lazy and let everything else fall through to eager.

Both snippets passed `shopify theme check` in a minimal theme. Try them on a development theme before you publish, and match `sizes` to your layout.

STEP 06

Measure before and after

Run Lighthouse before you touch anything and after each step, always with the same settings. This is how we measured our demo (Lighthouse installed with `npm install -g lighthouse`):

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

Swap in your page's URL. `--throttling-method=devtools` actually throttles network and CPU during the load; with the default simulation, the lazy-loading delay did not show up on our demo. Write down the median of three runs:

WHEN	LCP RUN 1	RUN 2	RUN 3	MEDIAN
Before				
No lazy				
With fetchpriority				
Size and format				

This is lab data: good for comparing changes. The real verdict comes from field data, in Search Console's Core Web Vitals report, a few weeks later.

FINAL

Checklist

- [ ] You know the LCP element on the home page and the product page.
- [ ] The hero has no `loading="lazy"` and no `data-src`.
- [ ] It has `fetchpriority="high"` and no other image does.
- [ ] It has `srcset`, a real `sizes`, `width` and `height`.
- [ ] A phone downloads a file for its width, as WebP or AVIF.
- [ ] On Shopify, lazy depends on `section.index > 3`.
- [ ] You measured before and after with the same settings.
- [ ] You scheduled a Core Web Vitals check in Search Console.