Melhorando o carregamento das imagens e SVGs

O primeiro e mais importante passo para melhorar a velocidade de carregamento de um site são as imagens. Se você ainda não sabe como medir a velocidade do seu site, o seguinte artigo pode te ajudar:

Medindo a velocidade do seu site

Se já trabalhou otimizando as imagens, talvez tenha interesse em ler sobre outras mudanças que podem melhorar a velocidade:

Melhorando o carregamento do Javascript

Melhorando o carregamento do CSS

Imagens

Formato

Com base em testes feitos em nossas lojas da Nuvem Shop, descobrimos que embora o formato PNG tenha uma boa qualidade, pode gerar um peso desnecessário, que pode ser reduzido em até 10 vezes quando é alterado para o formato JPG. Claro que se você precisa que a imagem tenha o canal alpha (utilize transparência), o JPG não vai poder ser uma opção, mas é importante considerá-lo.

Captura de tela 2018-12-06 as 14.24.54.png

Imagem PNG - 2,5 mb

Captura de tela 2018-12-06 as 14.24.54.jpg

Imagem JPEG - 181 kb

Produto da loja Argentina Unibow, onde a imagem passou de 2,5mb para 181 kb.

No exemplo de Unibow, o tempo de carregamento das imagens, após a mudança de formato e compressão

utilizando ferramentas como TinyPNG, baixou de quase **8.67 segundos para 2.07, são 6 segundos a menos!**

	Oportunidade	Poupança estimada
1	Publique imagens em formatos de última geração	8,67 s

	Oportunidade	Poupança estimada
1	Publique imagens em formatos de última geração	2,07 s

Antes (acima) e depois da mudança.

WebP

Como vimos no ponto anterior, ainda podemos melhorar o tempo de resposta de nossa loja usando formatos de próxima geração. WebP é um formato de imagem moderno desenvolvido pelo Google, que oferece compressão superior. É por isso que na Nuvemshop implementamos a conversão automática para o formato WEBP para banners e sliders.

Como uso o WebP na minha loja?

Para usar os diferentes tamanhos de imagens e WebP, você precisa ter seu código atualizado. Se você não abriu o FTP da sua loja, não precisa fazer nada, apenas lembre-se de fazer upload de imagens no formato jpg ou png. Caso contrário, precisamos que você use o maior número possível de tamanhos nos banners. Para isso você deve aplicar a extensão `settings_image_url()` indicando o valor da imagem desejada.

Por exemplo, se a imagem estiver assim:

```
<img data-src="" class="slider-image blur-up swiper-lazy" data-sizes="auto"/>
```

Podemos alterar, para que fique assim:

```
<img data-srcset='{{ slide.image | static_url | settings_image_url('large') }} 480w, {{ slide.image | static_url | settings_image_url('huge') }} 640w, {{ slide.image | static_url | settings_image_url('original') }} 1024w, {{ slide.image | static_url | settings_image_url('1080p') }} 1920w' data-sizes="auto" />
```

Tamanhos disponíveis

Tamanhos	Largura máximo
tiny	50px
thumb	100px
small	240px
medium	320px
large	480px
huge	640px
original	1024px

Tamanhos	Largura máximo
xlarge	1400px
1080p	1920px

Tenha em mente que, sempre respeitamos a proporção das imagens, então a altura será variável.

Para evitar perda de qualidade, apenas encolhemos as imagens, nunca as ampliamos.

SRCSET

Outra mudança chave é o uso de `srcset`, O que é? Basicamente imagens responsivas através de um atributo de HTML.

Adicionando o atributo `srcset` **além do** `src` **no tag** `img`, vai poder utilizar muitas urls da mesma imagem em diferentes tamanhos e o navegador vai carregar só a imagem adequada com base a, por exemplo, a largura da tela do dispositivo.

```
2    sizes="100vw"
3    srcset="http://placehold.it/200x150 200w,
4       http://placehold.it/400x300 400w,
5       http://placehold.it/600x450 600w,
6       http://placehold.it/800x600 800w,
7       http://placehold.it/1000x750 1000w,
8       http://placehold.it/1200x900 1200w"
9    src="http://placehold.it/300x150"
10   alt="">
```

Exemplo do artigo "How to srcset—responsive images"

Nas lojas utilizamos a implementação com base na unidade "w", na qual usamos a largura de cada imagem que temos. Tem muitas maneiras de utilizar `srcset`, recomendo ler mais sobre isso neste artigo.

Compartilho um exemplo de como é o código de lazyload + srcset para uma imagem de produto:

```
<div style="padding-bottom: {{ product.featured_image.dimensions['height'] / product.featured_image.dimensions['width'] * 100}}%;">
   <a href="" title="{{ product.name }}">
      <img alt="{{ product.featured_image.alt }}" data-sizes="auto" src="" data-srcset="{{ product.featured_image | product_image_url('small')}} 240w, {{ product.featured_image | product_image_url('medium')}} 320w" class="lazyload item-image img-absolute blur-up" />
   </a>
</div>
```

E é assim que seria para um banner:

```
<img src="{{ slide.image | static_url | settings_image_url('tiny') }}"
   data-srcset='{{ slide.image | static_url | settings_image_url('large') }} 480w, {{ slide.image | static_url | setti
ngs_image_url('huge') }} 640w, {{ slide.image | static_url | settings_image_url('original') }} 1024w, {{ slide.im
age | static_url | settings_image_url('1080p') }} 1920w'
   class="slider-image blur-up swiper-lazy" data-sizes="auto" alt="{{ 'Carrusel' | translate }}"/>
```

LazyLoad

Por que carregar todas as imagens se o usuário ainda não precisa vê-las? Podem alcançar isso utilizando da técnica de "LazyLoad" que basicamente carrega todas as imagens progressivamente quando o usuário está fazendo scroll ou abre uma tela ou pop-up que tem as imagens que não foram carregadas.

Desta forma, um enorme tempo é economizado até que o documento finaliza o carregamento das imagens. Por que carregar as imagens que estão no rodapé ou no pop-up no momento que o site é carregado?

Hoje existem dezenas de plugins de JS para utilizar LazyLoad, na Nuvem Shop utilizamos lazysizes. Alguns pontos sobre lazysizes:

- Não depende de jQuery, importante para carregar o código mínimo bloqueante (Javascript ou CSS)
- É fácil de utilizar, simplesmente adicionando a class "lazyload" na img, depois a url de uma imagem placeholder (ou pode ser a mesma imagem em baixa qualidade) é adicionada no atributo src e finalmente utilizando o atributo `data-src` (ou `data-srcset` se você precisa srcset) com a url da imagem final de boa qualidade. Uma vez que a imagem é carregada, a class "lazyload" vai mudar para "lazyloaded", você pode utilizar as classes para fazer mudanças no CSS uma vez que o carregamento da imagem finalizou.
- Se estão utilizando `srcset` é importante levar em conta as extensões respimg (para os browsers velhos como IE 9 e 11) e `bgset` caso que precisem utilizar uma imagem com background-image além do `srcset`.
- Outro ponto interessante é que o plugin não vai carregar as imagens que tenham ou estejam dentro de um elemento com `"display:none;"`.
- Se você precisa utilizar ainda mais funcionalidades do plugin, pode dar uma olhada as diferentes extensiões que tem

Por outro lado, uma das métricas de velocidade de carregamento das páginas da web é o LCP (Largest Contentful Paint), que mede o tempo em que o conteúdo visualmente maior, é exibido. Por esse motivo, **não é recomendado** aplicar *lazyload* as imagens que devem ser exibidas assim que a página for carregada (também chamada de "above the fold") porque provavelmente aumentará o tempo de LCP, diminuindo a pontuação geral de desempenho.

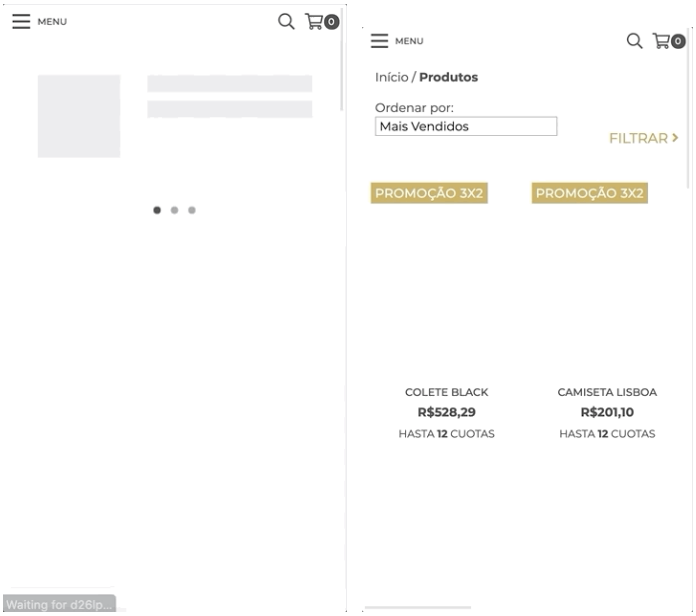
Por exemplo, no mobile, um dos maiores elementos que vemos frequentemente é o carrossel de imagens, então para a primeira imagem visível sugere-se deixar o "srcset" do conteúdo final.

Placeholders

Você tem LazyLoad, Parabéns! Mas, o que acontece até que a imagem final é exibida?

Muitas coisas podem ser feitas até ter a imagem final, em nossas lojas o que fizemos foi aplicar a técnica de LWIP (Low Quality Image Placeholders) carregando a imagem com um tamanho pequeno e um efeito de blur até que o lazy load exiba a imagem final.

Para utilizar LQIP, tem que aplicar no atributo `src` da tag `img` , a imagem em baixa qualidade e peso (na Nuvem Shop temos muitas verões da mesma imagem, a mais pequena com 50px de largura, os tamanhos que utilizamos são: tiny de 50px de largura, thumb de 100px, small de 240px, medium de 320px, large de 480px, huge de 640px e original de 1024px) com um CSS que aplica um width de 100% ao seu contêiner e logo quando a imagem é "lazyloaded" é substituida pela imagem de boa qualidade. No meio podem utilizar transições de CSS, nós aplicamos um efeito de blur animado.



O código fica mais o menos como o seguinte:

```
<img alt=”{{ product.featured_image.alt }}” data-sizes=”auto” src=”{{ product.featured_image | product_ima
ge_url(‘tiny’)}}” data-src=”” class=”lazyload item-image img-absolute blur-up” />
```

Onde você pode notar que utilizamos a class `lazysizes` para lazysizes, a class `blur-up` na transição de “imagem blured” a “imagem final” e os atributos src para a imagem placeholder em baixa qualidade (tiny) e `data-src` na imagem final (mais adiante a gente vai utilizar o mesmo exemplo mas com `srcset`).

O CSS para a transição é o seguinte:

```
.blur-up {
  -webkit-filter: blur(6px);
  filter: blur(6px);
  transition: filter .5s, -webkit-filter .5s;
}
.blur-up.lazyloaded {
  -webkit-filter: blur(0);
  filter: blur(0);
}
```

Utilizando a class “lazyloaded” pode ser de ajuda para exibir um ícone e então ocultá-lo quando a imagem foi carregada.

Ainda utilizando o código anterior o resultado não é o melhor, vai ter quebras visuais desde que a imagem pequena é exibida (ou um ícone) até que a imagem final seja carregada. Para evitar isso precisa saber o espaço que vai ocupar a imagem antes que seja carregada e para isso vai precisar conhecer as dimensões de alto e largura desde o backend.
Com essa informação você vai ter que fazer o seguinte:

```
<div style="padding-bottom: {{ product.featured_image.dimensions['height'] / product.featured_image.dimensions['width'] * 100}}%;">
   <a href="" title="{{ product.name }}">
      <img alt="{{ product.featured_image.alt }}" data-sizes="auto" src="" data-src="" class="lazyload item-image img-absolute blur-up" />
   </a>
</div>
```

Depois que a imagem seja carregada vai ficar como a seguinte imagem:

Vai ver duas coisas:

- Um estilo inline no contêiner da imagem, que basicamente é a conta: `alto/largura*100`. Essa conta vai dar uma porcentagem que vai utilizar com um `padding-bottom`, que vai "empurrar" o espaço que a imagem ocupará com a relação de aparência correta (de muita utilidade também em listagens tipo Pinterest). A conta vai dar algo como `padding-bottom: 133.68146214099%;`.
- Por outro lado você vai precisar que a imagem fique localizada bem dentro do contêiner sem que seja empurrada pelo `padding` adicionado. Para isso precisam utilizar a class `img-absolute` com o seguinte CSS:

```
.img-absolute {
  position: absolute;
  left: 0;
  width: 100%;
  height: auto;
  vertical-align: middle;
  text-indent: -9999px;
  z-index: 1;
}
```

SVGs

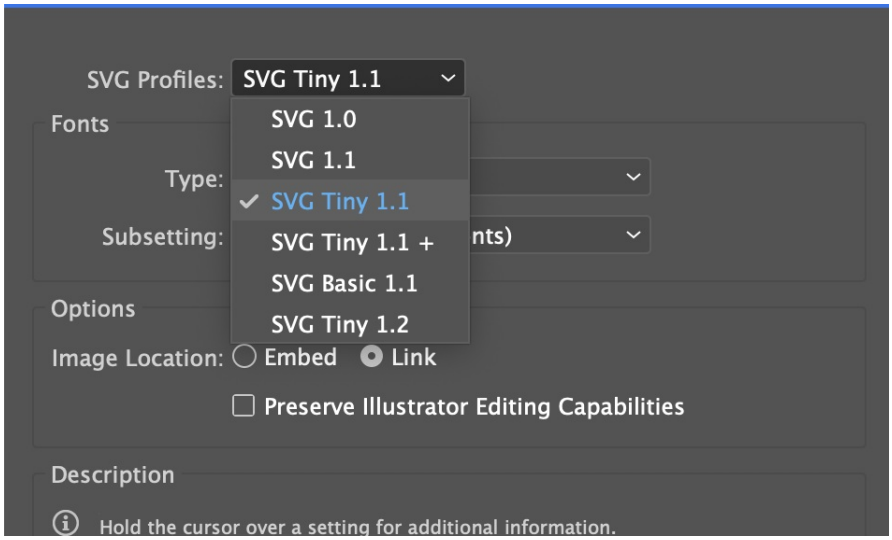
O SVG também é um formato gráfico como a imagem e mellhora a velocidade de carregamento já que é um vetor que não tem atraso na hora de ser “pintado” pelo navegador como acontece com as imagens onde tem que ser pintadas/carregadas pixel por pixel.

Nosso caso utilizamos SVGs para carregar todos os ícones críticos que precisamos que estejam visíveis nos primeiros segundos do carregamento do site: uma lupa de pesquisa, o carrinho de compras, o hamburger, etc.

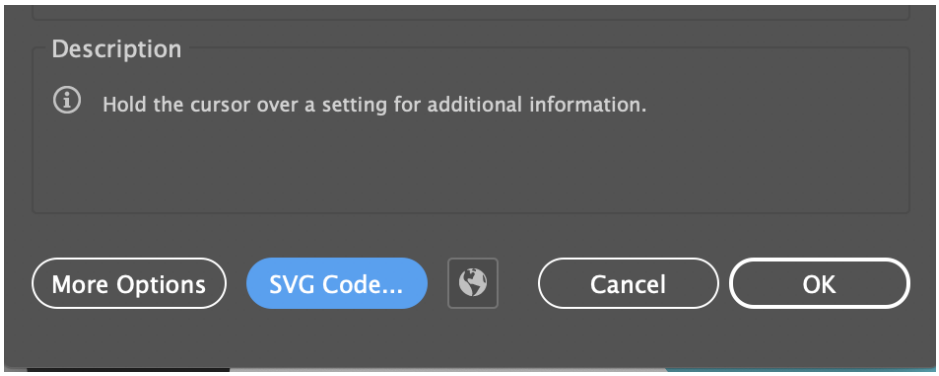
Deste jeito vai poder carregar os SVGs muito rápido de maneira inline e deixar o resto dos ícones com uma família tipográfica (nos utilizamos Font Awesome) e sejam carregados de forma assincrônica pelo CSS.

Para utilizar os SVGs tem que:

- Criar o SVG num software como pode ser Illustrator e descarregar SVG feito.
- No Illustrator na opção de "salvar como" tem que escolher o formato SVG Tiny 1.1.

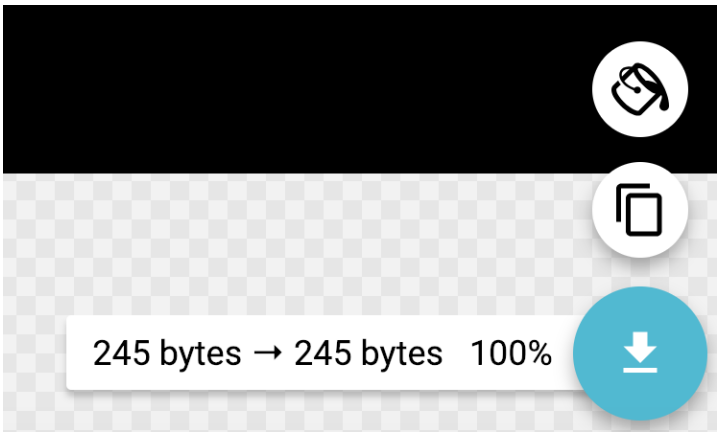


- Depois tem que escolher a opção código SVG.



- Finalmente tem que copiar e colar no site SVGOMG que vai ajudar a comprimir o arquivo ainda mais.

  Logo tem que fazer clique no ícone de "copiar"



Você já tem o código SVG pronto para ser colado onde quiser, é importante que ele fique _inline_ no layout para evitar uma _request_ desnecessária ao navegador. Na Nuvemshop, geramos um arquivo para cada SVG e depois importamos com o Twig usando um _include_ que basicamente faz a mesma coisa como se estivesse _inline_ no HTML.

O arquivo criado é um .tpl que usa o atributo {{svg_custom_class}}.

```
<svg class="{{ svg_custom_class }}" xmlns="http://www.w3.org/2000/svg" viewBox="0 0 38 32.5"><path d="
M18.5,28.5a4,4,0,1,1-4-4A4,4,0,0,1,18.5,28.5Zm11-4a4,4,0,1,0,4,4A4,4,0,0,0,29.5,24.5Zm5.44-3.44,3-13A2.5,2.5
,0,0,0,35.5,5H16a2.5,2.5,0,0,0,0,5H32.36l-1.85,8h-17L9.94,2A2.49,2.49,0,0,0,7.5,0h-5a2.5,2.5,0,0,0,0,5h3L9.06,2
1a2.49,2.49,0,0,0,2.44,2h21A2.51,2.51,0,0,0,34.94,21.06Z"/></svg>
```

Usamos esse atributo para passar a classe que precisamos para incluí-lo:

```
{% include "snipplets/svg/cart.tpl" with {svg_custom_class: "icon-inline icon-2x"} %}
```

Usamos um CSS que define tamanhos diferentes para os SVGs, muito semelhante ao que o Font Awesome usa.

```
.icon-inline {
  display: inline-block;
  font-size: inherit;
  height: 1em;
  overflow: visible;
  vertical-align: -.125em;
}


.icon-xs {
  font-size: .75em;
}
.icon-sm {
  font-size: .875em;
}
.icon-lg {
  font-size: 1.33333em;
  line-height: .75em;
  vertical-align: -.0667em;
}
.icon-2x {
  font-size: 2em;
}
.icon-3x {
  font-size: 3em;
}
.icon-4x {
  font-size: 4em;
}
.icon-5x {
  font-size: 5em;
}
.icon-6x {
  font-size: 6em;
}
.icon-7x {
  font-size: 7em;
}
```

```
.icon-8x {
  font-size: 8em;
}
.icon-9x {
  font-size: 9em;
}


.icon-inline.icon-lg{
  vertical-align: -.225em
}
.icon-inline.icon-w {
  text-align: center;
  width: 1.25em
}
.icon-inline.icon-w-1{
  width:.0625em
}
.icon-inline.icon-w-2{
  width:.125em
}
.icon-inline.icon-w-3{
  width:.1875em
}
.icon-inline.icon-w-4{
  width:.25em
}
.icon-inline.icon-w-5{
  width:.3125em
}
.icon-inline.icon-w-6{
  width:.375em
}
.icon-inline.icon-w-7{
  width:.4375em
}
.icon-inline.icon-w-8{
  width:.5em
}
.icon-inline.icon-w-9{
  width:.5625em
}
.icon-inline.icon-w-10{
  width:.625em
}
.icon-inline.icon-w-11{
  width:.6875em
}
.icon-inline.icon-w-12{
  width:.75em
}
.icon-inline.icon-w-13{
  width:.8125em
}
.icon-inline.icon-w-14{
  width: .875em
```

```
  width:.875em
}
.icon-inline.icon-w-15{
  width:.9375em
}
.icon-inline.icon-w-16{
  width:1em
}
.icon-inline.icon-w-17{
  width:1.0625em
}
.icon-inline.icon-w-18{
  width:1.125em
}
.icon-inline.icon-w-19{
  width:1.1875em
}
.icon-inline.icon-w-20{
  width:1.25em
}
.icon-spin{
  -webkit-animation:icon-spin .5s infinite linear;
  animation:icon-spin .5s infinite linear
}
@-webkit-keyframes icon-spin {
  0% {
    -webkit-transform: rotate(0);
    transform: rotate(0)
  }
  100% {
    -webkit-transform: rotate(360deg);
    transform: rotate(360deg)
  }
}


@keyframes icon-spin {
  0% {
    -webkit-transform: rotate(0);
    transform: rotate(0)
  }
  100% {
    -webkit-transform: rotate(360deg);
    transform: rotate(360deg)
  }
}
```

Que segue?

Se já pôde otimizar tudo o que tem a ver com imagens recomendamos continuar lendo as mudanças feitas em outras áreas:

Melhorando o carregamento do Javascript

Melhorando o carregamento do CSS