

# Performance Web II: Critical Path, HTTP/2 e Resource Hints: Aula 8 - Atividade 4

## Resource Hints | Alura

9-12 minutes

---

### Resource Hints: DNS- PREFETCH and PRECONNECT

Vamos tentar entender como funcionam os `rel` e `preloads`.

Queremos abordar também os *Resource Hints*. O primeiro *Resource Hint* que foi inventado foi o *DNS-PREFETCH* que permite pedir ao navegador para resolver o DNS de um certo recurso antecipando que precisaremos dele mais adiante. Podemos utilizar isso em nosso `index.html`, criando uma tag "Link" e usando:

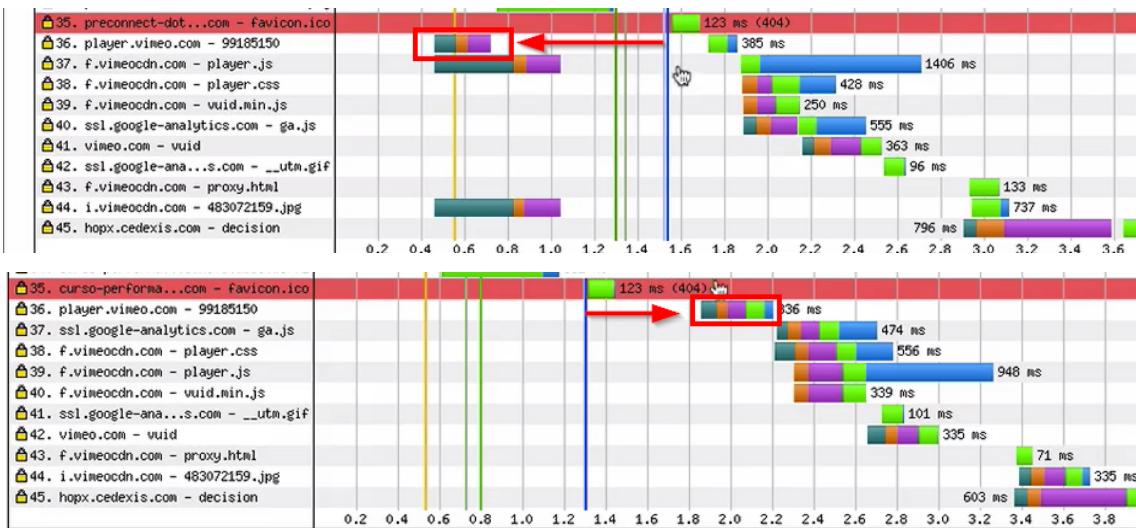
```
<link rel="dns-prefetch" href="/player.vimeo.com/">
```

Com isso, a resolução do DNS é antecipada para o começo. Quando ele fizer o *parcing* do HTML. A resolução do DNS é feita totalmente em *background* o que não significa que isso não travará a renderização ou o download das demais coisas. Também podemos fazer o mesmo com outras coisas e podemos utilizar esse recurso quantas vezes fizer sentido para o site.

Mostramos ele pois é o que possui melhor suporte, funciona em todos os navegadores. Foi um dos primeiros a ser inventado desses *Resource Hints*. Logo, se percebeu que podia-se dar passos além, não apenas resolver o DNS.

Um segundo *Resource Hint* é o *preconnected*. Além de antecipar o DNS podemos antecipar, também, a conexão. Para usar isso podemos escrever `link rel=preconnect`. Ele inclui a resolução do DNS, faz a conexão TCP e se for uma conexão segura ele inclusive faz a

negociação CSL para nós. Você pode se perguntar se isso realmente impacta em algo, repare nos gráficos. O primeiro diz respeito as mudanças que acabamos de fazer:



O que fizemos foi antecipar a conexão. Não podemos utilizar isso com diversas conexões, pois pode acabar atrapalhando as conexões que já temos. Podemos fazer isso com algum recurso adicional, por exemplo, um vídeo. Coisas que você gostaria de carregar mais rapidamente e que devem ser indicadas para o navegador.

## PREFETCH

O prefetch serve para passarmos para ele alguma "url" ou arquivo para que faça o download e guardemos isso no seu *cache*. O prefetch antecipa o download imaginando que esse download será utilizado no futuro. Ele serve para navegações secundárias, ele possui menos prioridade do que qualquer outro *request* que tenhamos feito na página, por isso, ele é indicado para ser feito na página seguinte.

Podemos antecipar o download do "categorias.js" digitando:

```
<link rel="prefetch" href="pagina-seguinte/categoria.js">
```

Assim, antecipamos esse download imaginando que a chance de ele clicar em alguma categoria da **Alura** é alta. Assim, já deixando isso no *cache*.

É preciso tomar cuidado, se abusarmos do uso dessas táticas é possível que começemo a baixar coisas que não são tão úteis, coisas a mais. Temos que utilizar esses recursos com certa esperteza. Outra

sugestão é também não fazer o prefetch de um vídeo de 2G, pois pode acontecer um potencial desperdício. O prefetch deve ser utilizado quando se visualiza uma chance alta de ser necessário.

O preload possui um uso similar, mas ele possui uma semântica distinta:

```
<link rel="preload" href="arquivo.css">
```

A diferença é que o prefetch possui uma prioridade baixa, está direcionado a uma navegação futura. O preload está direcionado para o momento, para a navegação atual.

O preload pode ser utilizado quando temos um carrossel de imagens e vários banners passando, colocamos o primeiro banner na imagem, mas sabendo que devemos antecipar o segundo banner, podemos pedir que o segundo banner seja antecipando o seguinte, pois na hora que o carrossel rodar o segundo banner já irá aparecer. Digitaremos o seguinte nesse caso:

```
<link rel="preload" href="banner2.jpg">
```

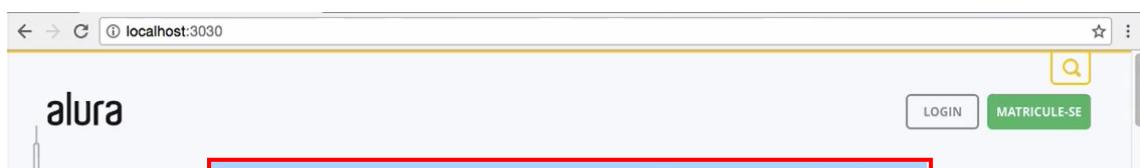
Já utilizamos o preload antes com coisas que queríamos que tivessem alta prioridade. Não poderíamos, nesse caso, usar o prefetch, pois desejamos uma prioridade alta.

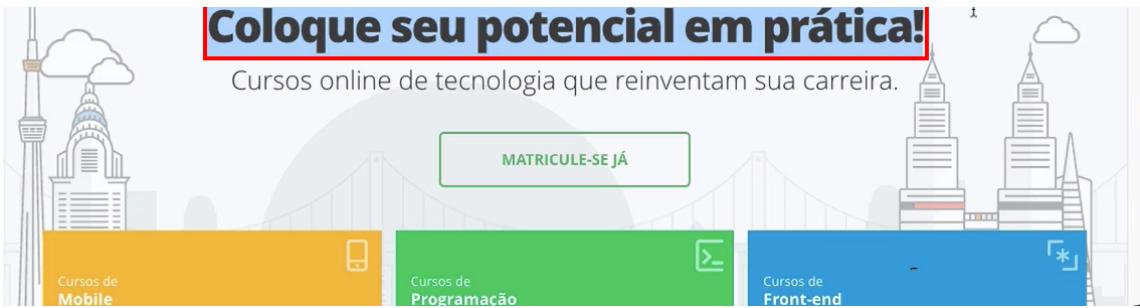
Imagine que temos diversos preloads possíveis, para saber qual deve ser o preload a ser utilizado devemos introduzir um atributo, o as, ele na verdade comunica a prioridade para o download do navegador:

```
<link rel="preload" href="arquivo.css" as="style">
```

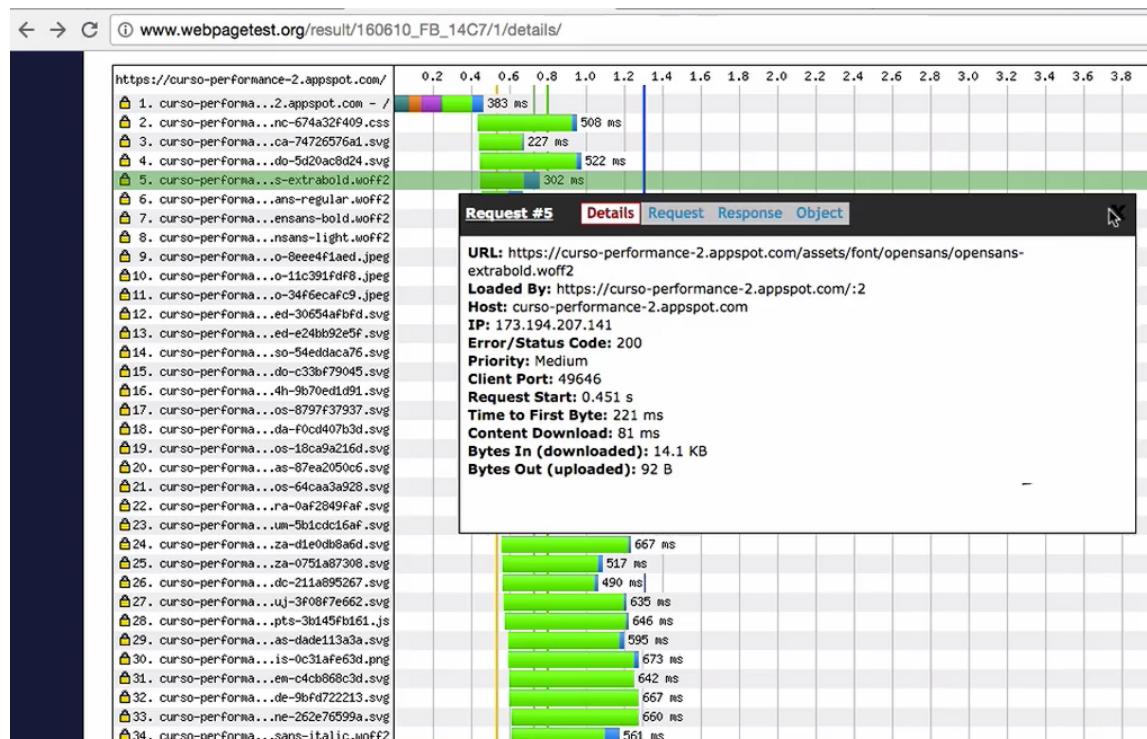
Ele lê o seguinte interpretando que o style possui prioridade, se fosse as="image" o navegador lê isso como algo com menor prioridade, pois sabe que isso é uma característica das imagens. Se você deixar sem o as isso indica que a prioridade é baixa.

Vamos pensar onde caberia o uso do preload no site, um recurso que costuma atrasar bastante a renderização do site é o seguinte:





A fonte! As fontes são alguns dos recursos que mais tardam em serem baixados na página, podemos, inclusive, reparar pela faixa em azul no gráfico "WaterFall":



Vamos relembrar, as fontes são declaradas no CSS:

```

index.html      font.css      Resource Hints
1 @font-face {
2   font-family: 'Open Sans';
3   src: local('Open Sans Light'), local('OpenSans-Light'),
4        url('assets/font/opensans/opensans-light.woff2') format('woff2'),
5        url('assets/font/opensans/opensans-light.woff') format('woff'),
6        url('assets/font/opensans/opensans-light.ttf') format('truetype');
7   font-style: normal;
8   font-weight: 300;
9 }
10
11 @font-face {
12   font-family: 'Open Sans';
13   src:
14     local('Open Sans'), local('OpenSans'),
15     url('assets/font/opensans/opensans-regular.woff2') format('woff2'),
16     url('assets/font/opensans/opensans-regular.woff') format('woff'),
17     url('assets/font/opensans/opensans-regular.ttf') format('truetype');
18   font-style: normal;
19   font-weight: 400;
20 }
21

```

E a "url" dessa fonte só será baixada na hora em que ela for realmente utilizada . O navegador adia o download da fonte o máximo possível.

Ele precisa primeiro baixar o CSS, "parsear" o CSS, baixar o HTML "parseá-lo" e encontrar algum nó que use a fonte e, então, inicia o *download*. E isso, pode ser demorado. E, assim, podemos fazer o *preload* disso, escrevendo:

```
<link rel="preload" href="assets/font/opensans-extrabold.woff2" as="font" type="font/woff2">
```

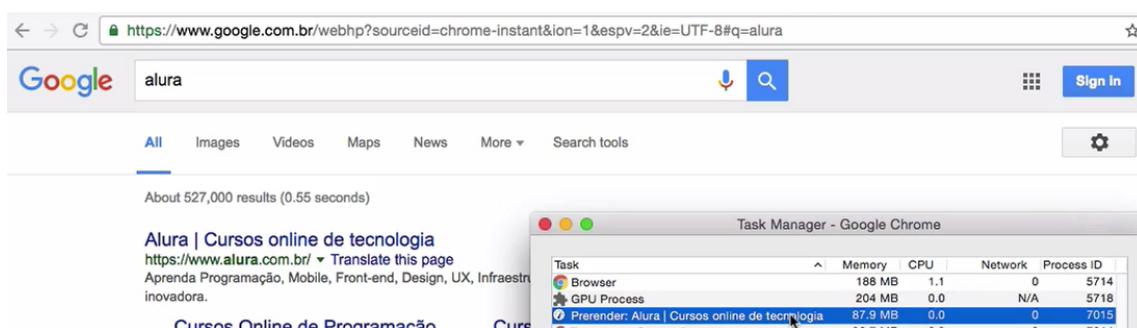
Agora, faremos esse download mais rapidamente. O *type="font/woff2"* serve para indicar que o download só vai ser feito se o navegador entender o woff2 e você pode usar o woff1 para os navegadores que entendem o woff1. O *preload* é bem útil para baixar recursos que são tardiamente descobertos, seja por ser assíncrono, secundário, por algum motivo que não é possível que ele seja descoberto logo de cara.

## PRERENDER

Para fechamos o tema de *Resources Hint* vamos finalizar abordando o prerender.

Isso pode parecer estranho, mas na verdade já utilizamos o prerender antes. Imagine, quando realizamos uma busca no **Google** ele nos mostra os resultados relacionados as palavras-chaves que usamos. Se buscarmos **Alura** a chance de clicarmos no primeiro site que aparece é muito grande e existe tanto essa certeza que o primeiro link já começa a ser renderizado antes mesmo de ser clicado, é isso que faz o prerender.

No **Chrome** se formos em "More Tools > Tool Manager" podemos verificar todas as abas que estão abertas e nessa janela podemos visualizar a aba "Prerender". Essa página ja foi renderizada. É como se tivesse uma aba escondida:



No momento em que clicamos no primeiro link que aparece na lista ele abre quase que instantaneamente. Isso acontece quando fazemos uma busca e o **Google** acredita que a possibilidade de ter acertado é grande. A ação do usuário é antecipada.

Para fazer isso é bastante fácil. Vamos voltar ao código, na "index.html" e acrescentamos o link `rel="prerender"` e a página que deve ser pré-renderizada `href="pagina.html"`:

```
<link rel="prerender" href="pagina.html">
```

É importante usar esse recurso se você tiver uma razoável certeza de que isso vai ser utilizado, isso é muito útil quando temos os casos de *work flow*. Por exemplo, no caso da leitura de um texto que possui mais de uma página, é provável que a notícia siga sendo lida, a probabilidade da pessoa continuar até as próximas páginas é grande.

No site do **Alura** existe um cenário interessante, no final da página existe um campo para se cadastrar e receber novidades e lançamentos:

Quando terminamos de preencher com nossas informações esse campo e damos um "ok" somos redirecionados para uma página que diz "Obrigada!":



A implementação disso é curioso, no "index.html" temos o *input* com o cadastro:

```
681     </ul>
682   </div>
683
684   <div class="footer-column">
685     <div class="footer-newsletter">
686       <h2 class="footer-newsletter-titulo">Receba Novidades e Lançamentos</h2>
687
688       <div class="footer-newsletter-form">
689         <input type="email" class="footer-newsletter-input" placeholder="Digite seu e-mail" /><button class="footer-newsletter-button buttonForm">OK</button>
690     </div>
691   </div>
692   <div class="footer-parceiros">
693     <h2 class="footer-parceiros-titulo">Parceiros</h2>
694     <a class="footer-parceiros-galandra" href="http://www.galandra.com.br">
695       
696     </a>

```

E na hora que clicamos nisso podemos observar que ele não dispara nenhum formulário, isso foi implementado utilizando o *javascript*. Isso está no "footer.js":

```
1 settimeout(function() {
2   // Your code here
3 }
```

```

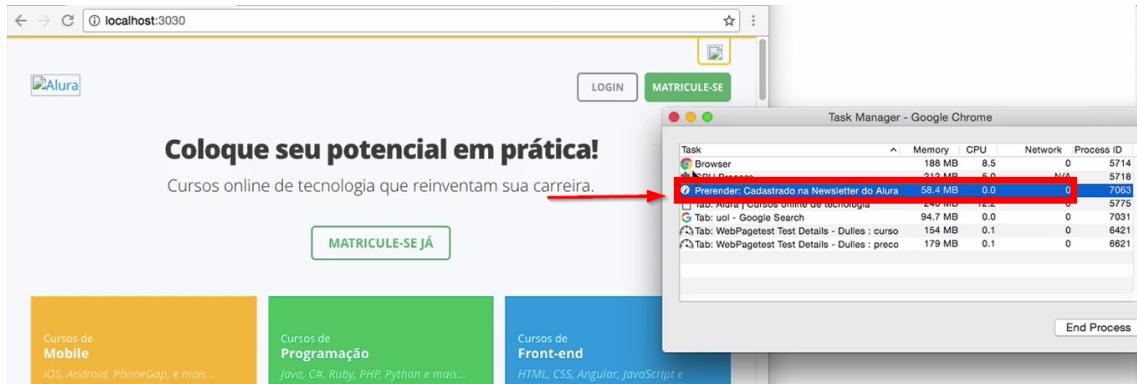
3   var newsletterButton = document.querySelector('.footer-newsletter-button');
4   var inputEmail = document.querySelector('.footer-newsletter-input');
5
6   if (newsletterButton) {
7     newsletterButton.onclick = cadastraNewsletter
8   }
9
10
11
12 // quando clicar no botão, valida o email,
13 // chama API e redireciona pra pagina de confirmacao
14 function cadastraNesletter() {
15   if (valida()) {
16     chamaAPI(inputEmail.value, function() {
17       location.href = 'cadastrado.html';
18     });
19   }

```

A regra é que quando a pessoa clicar no botão de *Newsletter* ele cadasra a newsletter fazendo uma validação, chamando a API e ao terminar de fazer isso somos redirecionados para a página do cadastrado. É uma redirect. Esse cenário é um forte candidato para fazermos um prerender. Escreveremos o seguinte no "index.html":

```
<link rel="prerender" href="cadastrado.html">
```

Fazendo um gulp e carregando a página o que veremos é um prerender do "cadastrado":



Outra técnica que podemos utilizar, já que o usuário só vai chegar ao prerender se ele preencher o campo das *Newsletter* é criar o link `rel="prerender"` de maneira dinâmica. Imagine que o usuário está na página e começa a preencher o campo, nesse caso, a chance de ele apertar o "ok" é altíssima. Podemos pegar esse evento para disparar o download. Vamos no "footer.js" e diremos que quando a pessoa der foco, aí se criará o elemento do prerender:

```

inputEmail.onfocus = function() {
  var prerender = document.createElement('link');
  prerender.rel = `prerender`;
  prerender.href = 'cadastrado.html';
}

```

```
document.head.appendChild(prerender);  
  
}
```

O ponto é tentar descobrir qual o melhor momento para fazer isso. E isso dependerá do cenário. Vamos testar isso! Antes ele dava prerender apenas de abrirmos a página, agora, ele disparará o prerender apenas quando dermos foco no campo do e-mail:

The image displays two screenshots of a web browser (Google Chrome) running on localhost:3030. Both screenshots show the Alura landing page with various course statistics: 208 cursos, 1.638 aulas, and 7.391 exercícios. A 'RECEBA NOVIDADES E LANÇAMENTOS' form is present, featuring an 'e-mail' input field and an 'OK' button.

In the top screenshot, the Task Manager window is open, showing a list of tasks. One task, 'Prerender: http://localhost:3030/cadastrado.h...', is highlighted with a red box. This indicates that prerendering occurred before the user focused on the email input field.

In the bottom screenshot, the user has focused on the 'e-mail' input field. The Task Manager window is again open, and the same prerender task is highlighted with a red box, demonstrating that prerendering is triggered only when the input field gains focus.

Task	Memory	CPU	Network
Browser	189 MB	3.8	0
GPU Process	212 MB	0.0	N/A
Tab: Alura   Cursos online de tecnologia	284 MB	10.0	36.6 s
Tab: uol - Google Search	94.7 MB	0.0	0
Tab: WebPageTest Test Details - Dulles : curso	154 MB	0.1	0
Tab: WebPageTest Test Details - Dulles : preço	179 MB	0.1	0
Prerender: http://localhost:3030/cadastrado.h...	29.7 MB	0.0	