08

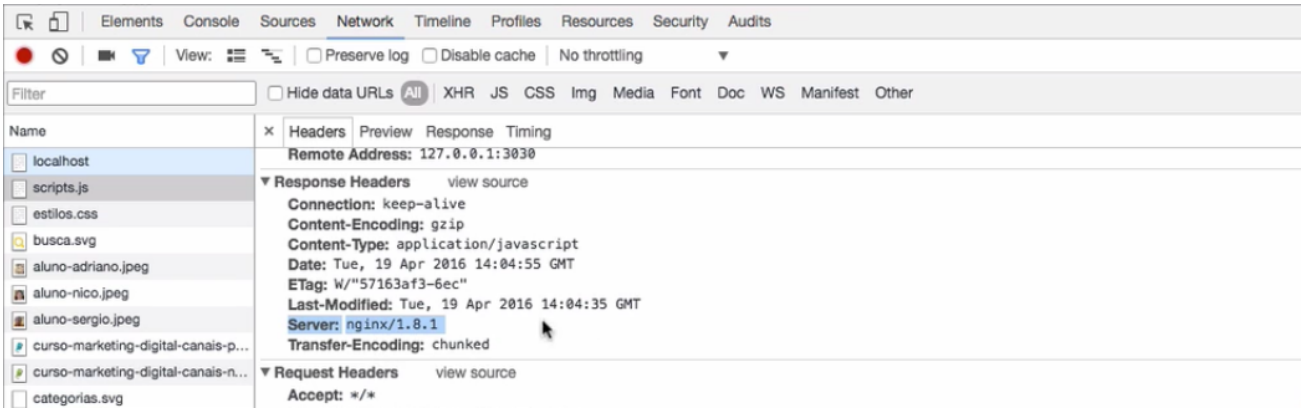
Transcrição das aulas

No mundo da performance, *request* bom é *request* não feito, por esse motivo fizemos diversas otimizações. Será que não conseguimos ir mais além?

Será que se continuarmos a acessar nossa página, os ícones mudam? E, se houver mudança, será que é frequente?

Estamos tentando chegar no tópico de *cache*. Por padrão, o navegador não faz *cache* e isso ocorre por que podem ocorrer alterações frequentes naquilo que será cacheado, assim, imagine a seguinte situação: o navegador faz *cache*, o site sofre alterações, e o navegador cacheia o conteúdo antigo, assim, quando o usuário acessa ele observará o site antigo.

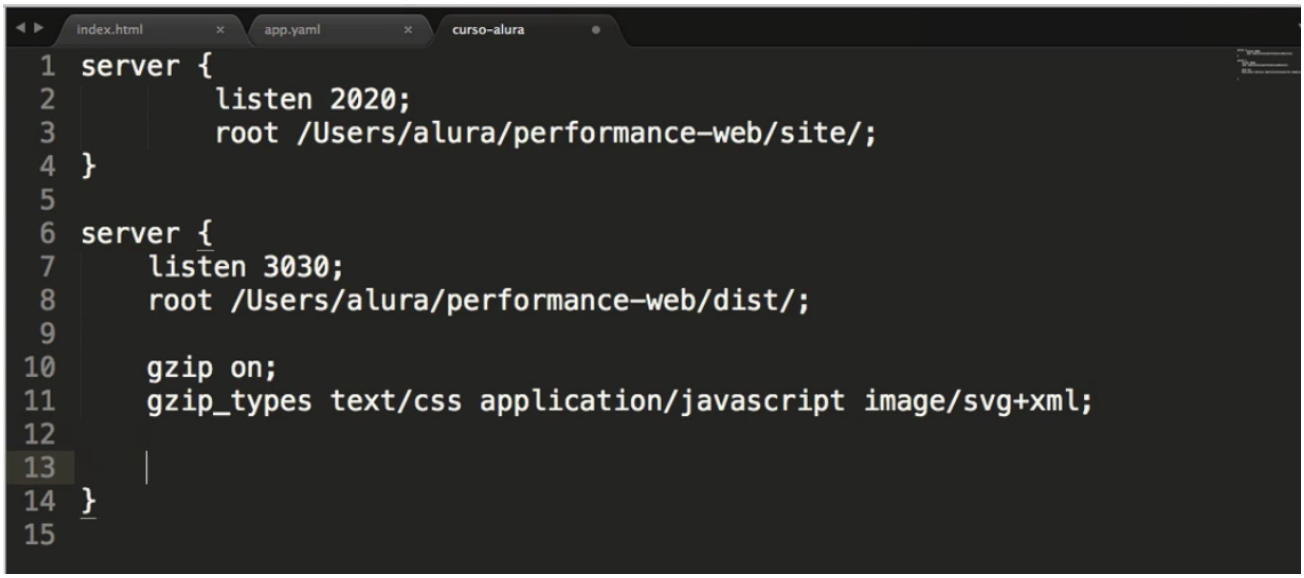
O navegador evita realizar o *cache* a não ser que o servidor instrua o navegador que determinado recurso é "ok" ser cacheado. Esse tipo de configuração fazemos no momento do servidor. O problema é que isso muda de servidor para servidor. A configuração terá que ser feita no momento que o servidor for configurado. Podemos fazer isso através de *headers* de *http*. Quando clicamos na aba *networking*, ele lista os recursos que baixamos e podemos clicar em um deles que ele mostra na aba *headers* todos os cabeçalhos que são enviados tanto no *request* quanto na resposta. Temos aqui diversas informações a respeito da *url*:



A ideia é que podemos adicionar no cabeçalho da resposta outros aspectos, podemos adicionar por quanto tempo o navegador pode cachear o recurso. Esse cabeçalho se chama *expires*.

O que temos que fazer?

Temos que configurar o servidor. Como estamos utilizando o *nginx* vamos configurar diretamente. Vamos em nossa pasta "curso-alura":

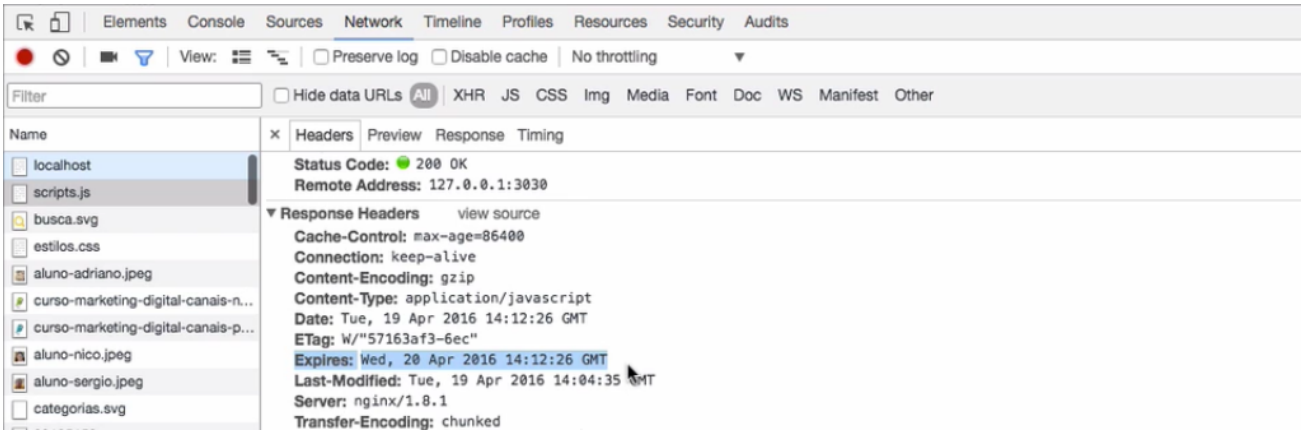


A única coisa que tinhamos habilitado aqui foi o *gzip* e agora vamos habilitar o *expires*. Para isso utilizamos um comando bastante simples, o *expires* e digitamos a quantidade de tempo que queremos que essa habilitação dure. O "1y" refere-se a 1 ano, o "1d" a 1 dia e assim por diante. Digitaremos:

```
expires 1d
```

Isso significa que ele vai adicionar, no servidor, o cabeçalho do *response* do cliente. E após termos feito isso vamos recarregar em nosso terminal escrevendo `reload -s reload` . E vamos acessar, novamente, o navegador.

O que veremos de diferente?



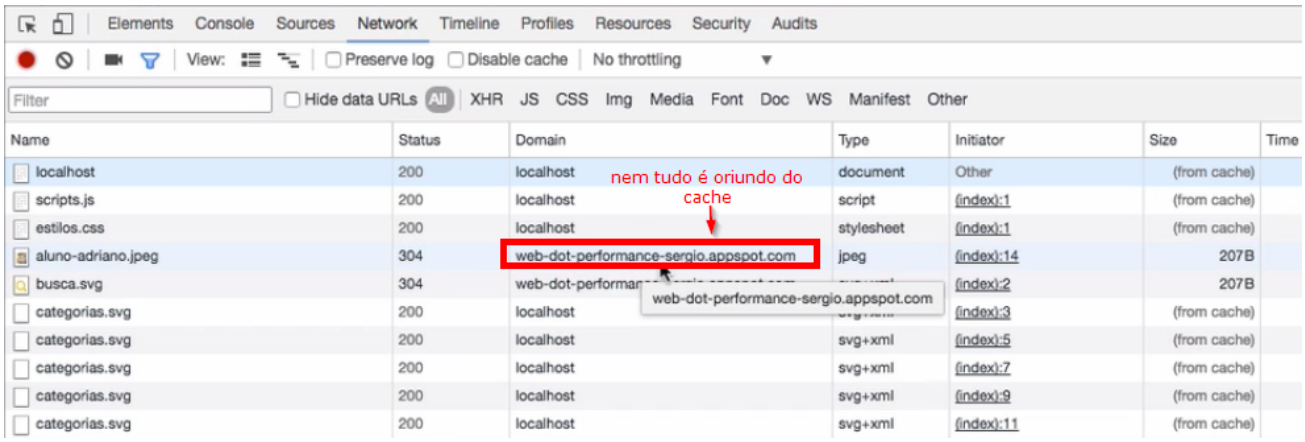
Bom, se observarmos o mesmo arquivo de antes podemos observar algumas diferenças. Por exemplo, temos um *expires*, até quando esse arquivo se manterá vigente, isto é, guardado em *cache*. Podemos reparar que estamos acessando a página em 19 de abril e estamos vendo que o *expires* está configurando para até 20 de abril, tendo duração de um dia, conforme o que tínhamos escrito.

Vamos fazer um teste, vamos atualizar nosso site, mas, atenção, que isso não significa que ele vai baixar as coisas do *cache*. Veremos que ele irá, novamente, no *local host*. Toda vez que o usuário atualiza o navegador, depende na verdade do navegador, ele tenta verificar se não tem atualizações. Para simular uma navegação no *cache* temos que simular uma navegação no usuário. Tipo, clicar em um link. Escondemos um link no logo do Alura, assim, ao clicar nisso faremos uma navegação para a própria *home*.

Fazendo isso e continuando com o *dev tools* aberto podemos ver na coluna *size* que os mesmos arquivos foram cacheados. Nessa coluna teríamos, normalmente, indicado o tamanho da rede utilizada e como ele foi cacheado, não

teremos o tamanho da rede. Ele nos diz que usa o *cache* local para baixar o arquivo economizando, assim, diversos *kbytes*.

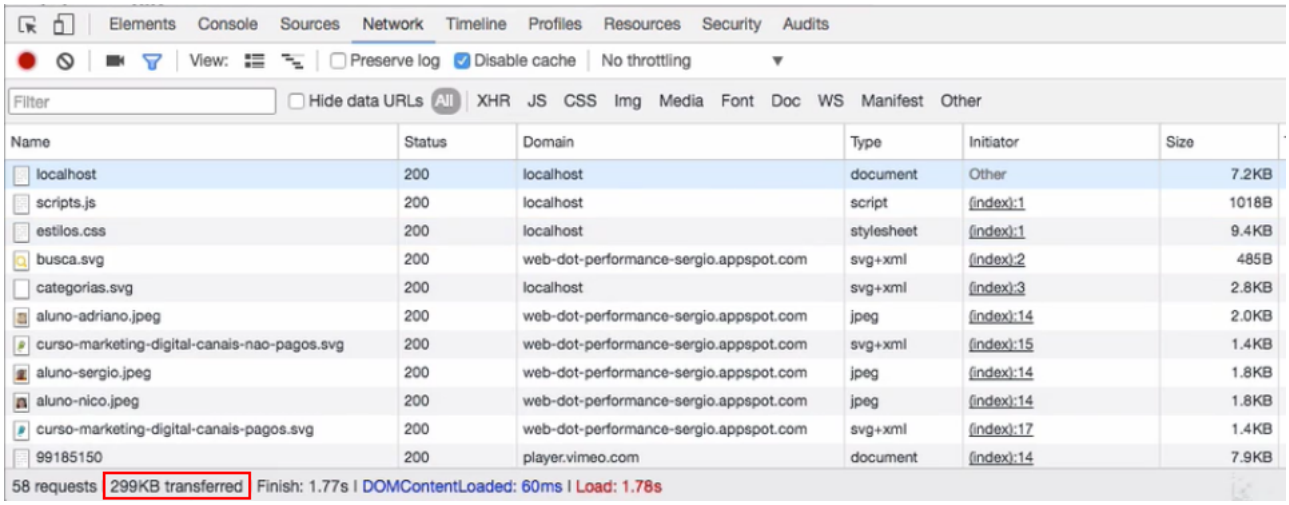
Se repararmos bem, nem tudo tem sua origem no *cache*:



Na verdade nós usamos coisas do domínio paralelo e nele nós não configuramos os *headers*.

Tem alguns detalhes que são importantes: Como testamos se virá sempre do *cache*?

Por exemplo, olharemos lá em baixo do *dev tools* e veremos que 15 *kbytes* foram transferidos, mas na verdade, sabemos que é muito mais. Lembra-se da opção *disable*? Se marcarmos essa opção, não interessa o que fizermos agora, sempre que estivermos com o *dev tools* aberto o *cache* estará disabilitado, então, ele sempre vai baixar os 300 KB repetidamente.

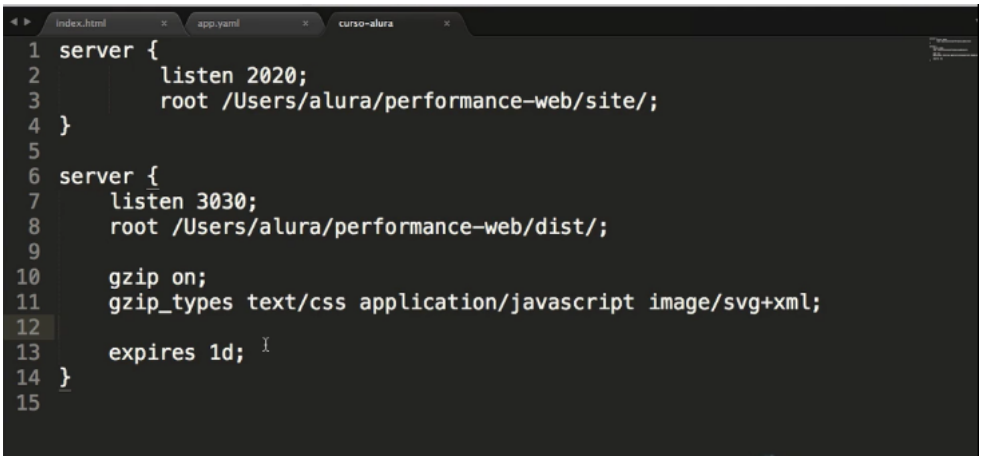


Temos ainda algumas outras coisas importantes que precisamos pensar.

Por exemplo, o *script* é algo que muda com pouca frequência, então, deixar ele com um *cache* de um dia não parece uma opção ruim. O "estilo.css" também não parece ser uma opção ruim. Agora, o *html* seria uma boa?

Ele também foi colocado *header* de *cache*. Mas não é muito boa ideia deixar o *html* cacheado, pois, é algo que alteramos com mais frequência, por exemplo, introduzindo as novidades do dia. Imagine o caso de um *html* de um blog ou algo dinâmico, não são casos que devem ser cacheados.

Vamos reparar que na hora que fizemos a alteração no *nginx* nós digitamos `expires 1d` que indica que todos os recursos do servidor estejam com *expires 1d*.

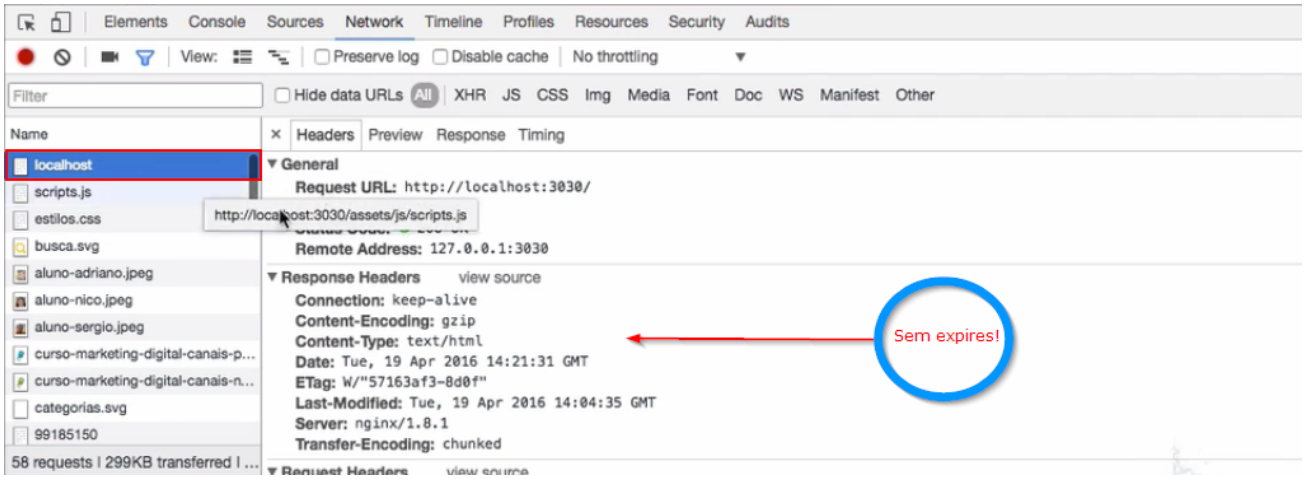


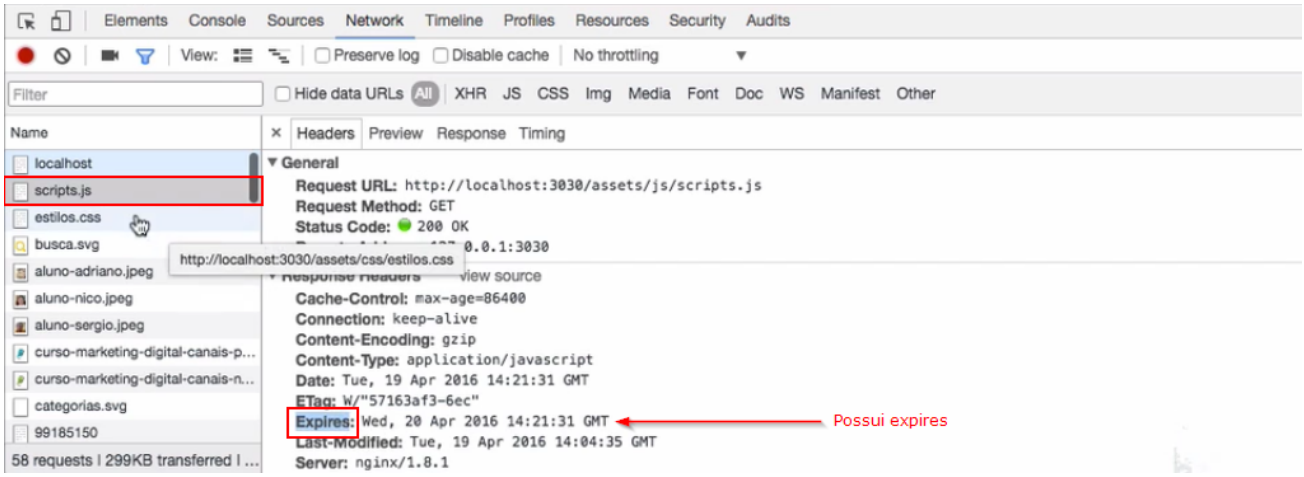
Mas, não é bem isso que queremos. Queremos deixar o `expires`, na verdade, apenas para os arquivos da pasta "assets" que são os arquivos mais estáticos (css, imagens, javascript e etc). No caso do *nginx* é muito fácil fazer isso, é só colocar um bloco `location /assets` e colocar a configuração do `expires` dentro disso.

```
server {
    listen 3030;
    root /Users/alura/performance-web/dist/;

    gzip on;
    gzip_types text/css application/javascript image/svg+xml;

    location /assets {
        expires 1d;
    }
}
```

O que estamos dizendo ao fazer isso é que nesse servidor nem tudo tem `expires`, apenas o que estiver na localização `assets`.

Vamos dar um *refrash* no navegador e vamos observar os *headers*. O do *html* não possui mais o `expires`, mas o "script.js", o "estilo.css" esses, ainda possuem. Conseguimos fazer o que efetivamente queríamos.





Outra coisa que gostaríamos que fosse percebida quando adicionamos o *header expires*, também adicionamos outro *header*, o *cache control*, que possui um atributo *max age*, que significa que ele deve ter um *cache* até daqui um dia. Então, no *cache control*, o tempo deve ser o mesmo que no *expires*. O próprio *nginx* faz isso por nós.

Por que estamos falando disso?

O *cache control* possui mais algumas configurações interessantes. Ele permite além de dizer o quando aquele arquivo expira também dizer quem pode cachear isso por nós. Por padrão, o *cache* foi feito para o navegador pode cachear. Então, acessamos um recurso e o navegador cacheia. Mas a ideia é que entre o navegador e o servidor, existem diversos intermediários na rede.

Talvez, se você trabalha em uma grande empresa, você tenha um *firewall* interno na empresa, ou um *proxy* que pode fazer *cache*. Podemos ter *proxys* em diversos níveis desse tráfego de rede, entre o servidor e o *browser*. A ideia é que se temos um recurso que pode ser cacheado pelo navegador, as vezes esse recurso também pode ser cacheado por outros intermediários, como os *proxys*. O que pode ser uma boa ideia.

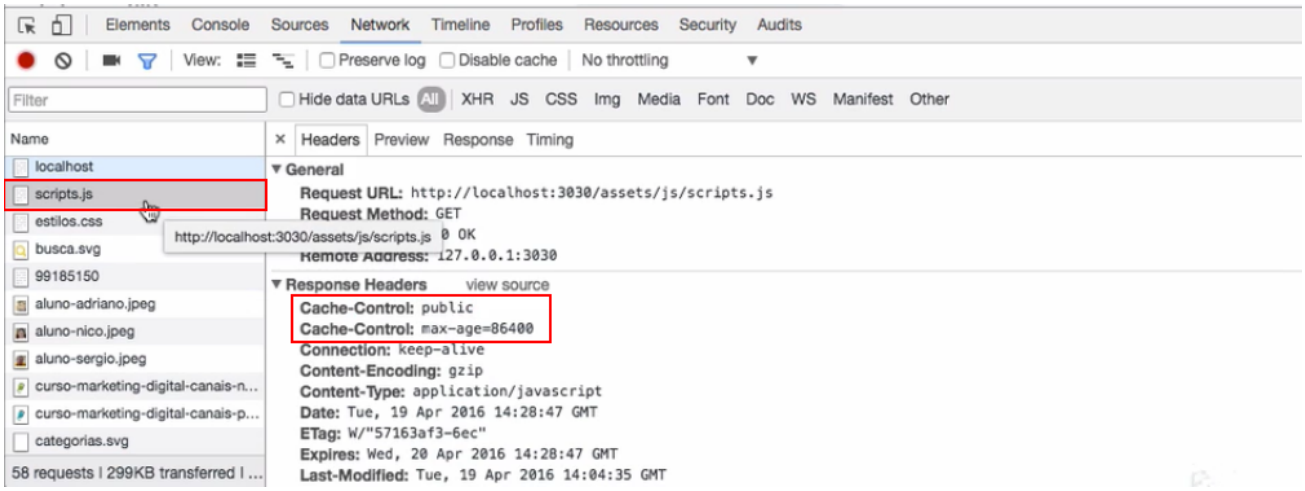
Por que estamos falando disso?

Podemos ir ao nosso servidor e adicionar um *header* `Cache-Control` com valor `public` :

```
location /assets {
    expires 1d;
    add_header public;
}
```

O que significa um *header* com valor público?

Vamos dar um `reload` e ir em nosso navegador. Vamos observar o *dev tools*. Podemos reparar que além do nosso recurso possuir um *cache-control* com um período de expiração, temos também um `Cache-control` que é `public` .



O `public` indica que qualquer intermediário pode cachear esse recurso. O que é muito interessante pois a ideia é que o intermediário mais próximo dele é que vai servir esse arquivo a primeira vez que ele baixar.

Mas em um caso de um recurso que muda de usuário para usuário, então, intermediários não podem cachear. Uma alternativa é mudar para `private` que significa que apenas o usuário final poderá cachear esse recurso.

Em geral, para *script*, imagens e outros, deixamos *cache-control* no modo `public` o que indica que qualquer um pode fazer o cacheamento.

Agora, que já discutimos a questão de como usar o `expires` , vamos discutir qual seria o tempo ideal para deixar os recursos cachearem. Queremos que nossos recursos sejam o máximo possível cacheados e que um mínimo deles sejam baixados novamente.

Qual seria o problema de deixar o *expires* com um ano de duração? Se mudarmos o site, como o usuário saberá que mudou?

Se o navegador cacheia durante um ano e se mudarmos nossos *css*, *javascript* e etc eles não serão baixados em sua versão atualizada. Na prática temos um conflito, queremos deixar o *expires* alto para evitar que ele faça *download* da mesma coisa várias vezes, mas também queremos que se houver uma modificação, por exemplo, no *css*, que o usuário fique sabendo disso.

Como resolver esse problema?

Quando falamos para um navegador que temos um `expires` de um ano ele *cacheia* a *url* que foi apontada pelo *html*. Uma técnica que podemos utilizar é a seguinte, deixar o `expires` alto, como mais ou menos um ano, mas, quando formos fazer alguma alteração, por exemplo, no arquivo "base.css", podemos dar um *rename* e podemos atualizar isso colocando o *rename* do arquivo.

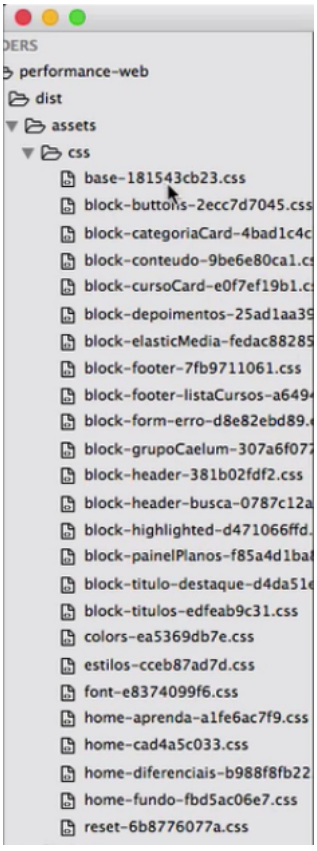
```
11
12      <!-- build:js assets/js/scripts.js -->
13      <script src="assets/js/menu.js"></script>
14      <script src="assets/js/busca.js"></script>
15      <script src="assets/js/detect.js"></script>
16      <script src="assets/js/footer.js"></script>
17      <!-- endbuild -->
18
19      <!-- build:css assets/css/estilos.css -->
20      <link rel="stylesheet" href="assets/css/reset.css">
21      <link rel="stylesheet" href="assets/css/base-atualizado.css">
22      <link rel="stylesheet" href="assets/css/colors.css">
23      <link rel="stylesheet" href="assets/css/font.css">
24
```

Como estamos juntando tudo no *build* temos que atualizar também o `build` com `estilos-atualizado.css` . Então, imagine que quando o usuário acessar o *html* novamente vai baixar o `estilos-atualizado` , ou seja, ele ignora o que foi cacheado e baixo o novo.

E se quisermos editar um novo arquivo? Teremos que ficar renomeando o `build` e inventando diversos nome, por exemplo, "mega-atualizado", "super-atualizado" e etc.

A ideia, entretanto, é que não façamos isso na mão. O que queremos fazer é apresentar outra maneira de realizar essa tarefa. Vamos desfazer tudo o que modificamos, pois queremos que tudo seja automatizado.

Até esse momento usamos o *gulp* para realizar automações. Usaremos um *pluggin* dele que pegará todos os *css* da pasta "assets" e irá renomear corretamente quando houver modificação. Esse plugin insere um código ao final dos arquivos e toda vez que houver alteração ele gera outra vez isso. Vamos no Terminal e rodamos o `gulp revreplace` .



Ele demora um pouco, mas após terminar veremos que todos os arquivos da pasta "css" tiveram seus nomes modificados para ter do lado deles um código. Esse código é de revisão, ele calcula um *hash* para cada arquivo e assim,

se o arquivo mudar, alterar-se-a o *hash* também.

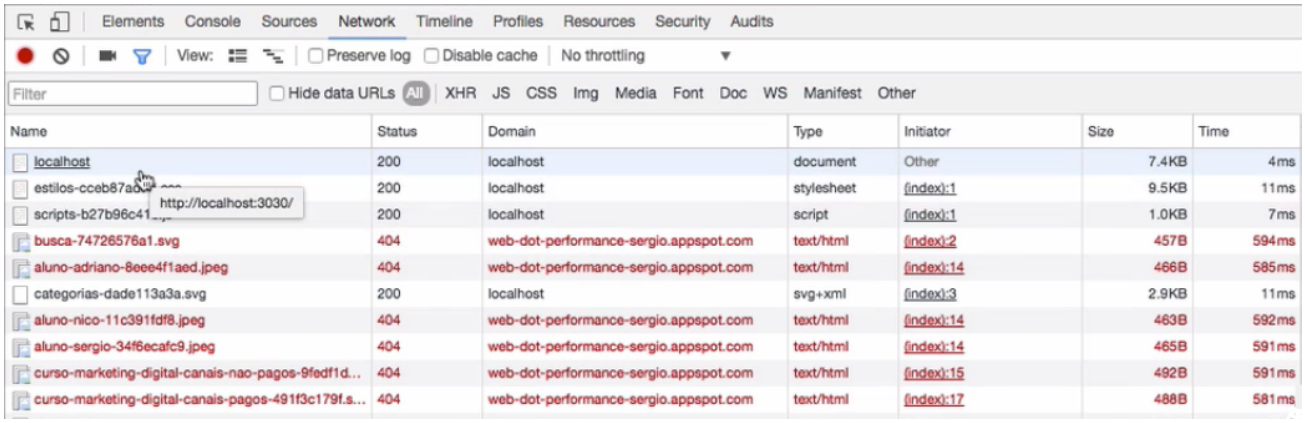
Ele renomeia todos os *css*, imagens e etc.

E como fazemos para renomear o *html* e chamar essa versão renomeada? Ele já faz isso por nós.

Lembre-se que o *html* está minificado então fica difícil visualizar, mas podemos pegar uma *tag* de *script* e veremos que ela está acompanhada do código e isso vale para tudo.

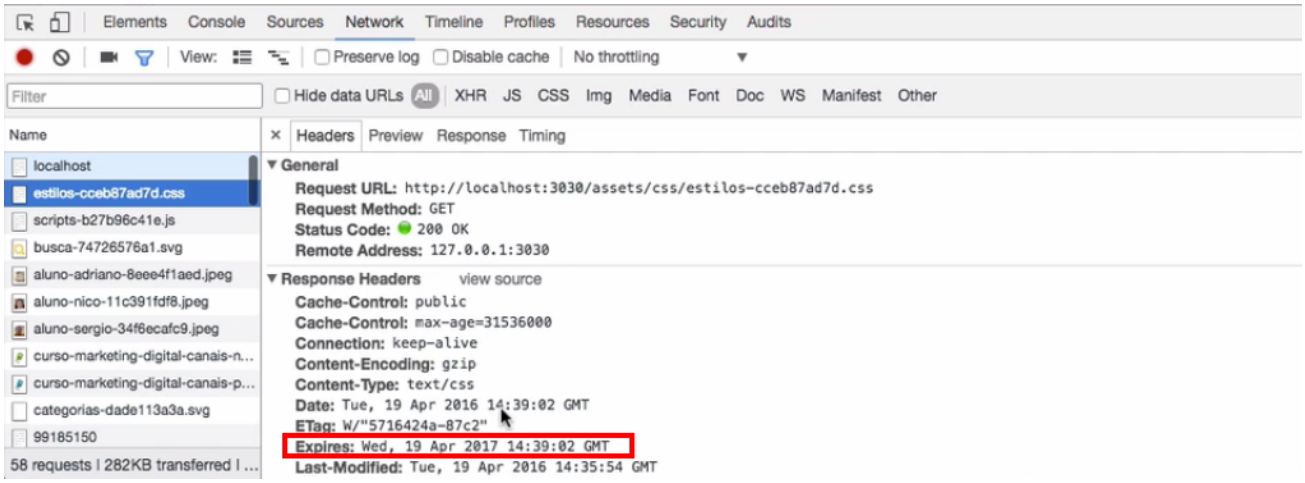
Esse é *pluggin* é muito útil. Se fizermos uma alteração na página, isto é, se mudarmos o conteúdo ele calcula um novo *hash* e coloca isso corretamente no *html*, nos dando mais segurança para utilizar um *expires* de um ano. Assim, podemos deixar o `expires` com uma validade alta pois sabemos que o pluggin irá gerar uma *url* diferente que será baixada pelo navegador.

Vamos observar no navegador:



Mas repare que temos diversas coisas em vermelho e isso acontece porque ele está tentando baixar conteúdos da nossa *home* secundária. Então, como renomeamos o arquivo, esses arquivos é como se não existissem mais. Temos que fazer o *deploy*. Estamos na nossa "app.yaml" e escrevemos `performance-sergio` no campo *applications* e quando abrir o aplicativo pedimos para fazer o *deploy*.

Agora, voltou a funcionar:

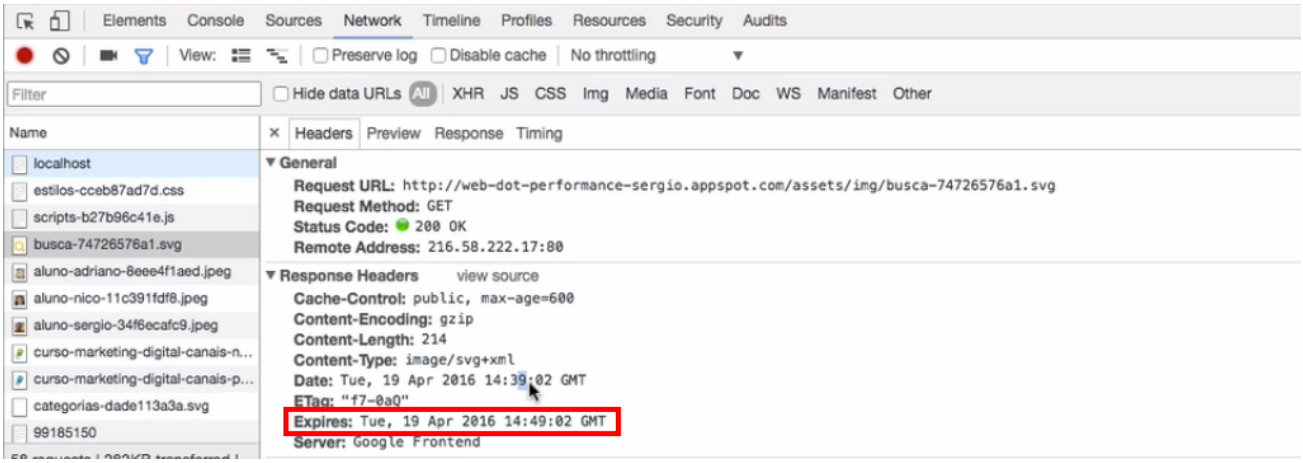


Se analisarmos o navegador veremos que temos o `expires` até o ano de 2017. Se olharmos o `Cache Control` teremos a mesma coisa.

Podemos fazer isso com o *html*? Todos os arquivos foram renomeados, com exceção do *index.html*. Faz sentido se tentarmos fazer isso?

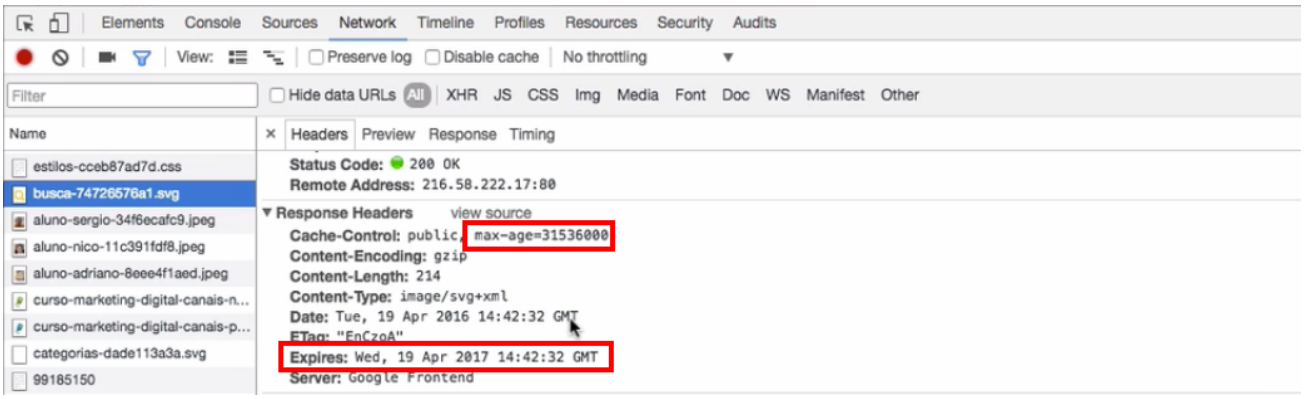
Não, pois o *html* é o ponto de entrada do usuário. A ideia é que a *url* seja estável, ou seja, não deve mudar a cada alteração, por isso, fizemos o `expires` apenas para os recursos do `assets` . Deixamos o *html* de lado, isto é, que ele seja expirado daqui um ano, se não, teríamos que gerar novos nomes o tempo todo e o usuário ficaria sem saber o que digitar.

Um último ponto que fizemos foi a configuração do *header* no *nginx* local. Já vimos que nem todos os recursos são baixados do *nginx* local, alguns são baixados de nosso link secundário. Lembrando que estamos utilizando o *app engine* e por padrão ele coloca um *expires*, que é bem baixo, na verdade,



E se quisermos alterar o *expires* no *app engines*?

Lembra-se do "app.yaml"? Temos nele a pasta "assets". Em baixo dela colocaremos um atributo `expiration` e a quantidade de tempo que desejamos que seja o tempo até expirar. Importante notar que a configuração funciona com dias. Após isso, fazemos o *deploy* e podemos conferir se funcionou:



Acabamos de mostrar dois servidores diferentes, o *google app engine* e o *nginx*. Como fazemos a configuração de expiração em *apache, is* e etc ? Geralmente essa configuração é um comando curto.

Recapitulando, o importante é colocar uma "expiração" longa para os recursos estáticos da página. É preciso tomarmos cuidado para que possamos atualizar isso depois e uma maneira de fazer isso é agregando um número de versão para esse recurso. Por exemplo, o *gulp* é capaz de fazer isso, mas se você quiser utilizar outra maneira pode pesquisar como fazer automação usando uma *url* diferente baseada no conteúdo dela.

É importante lembrar de se for inserir um *expires* alto, é preciso inserir algo para alterar a *url* se houver alteração de conteúdo.