Aula 26 – Estratégias de Cache

Parte 1: Cache no Cliente e no Gateway

No dinâmico universo do desenvolvimento de aplicações web, onde a expectativa por respostas instantâneas é a norma, a performance não é apenas um diferencial, mas uma exigência fundamental. Imagine um usuário tentando acessar um site que demora a carregar; a frustração é quase imediata, e a chance de ele abandonar a página e procurar um concorrente aumenta exponencialmente. Essa realidade se intensifica em arquiteturas modernas, como microsserviços e serverless, onde a comunicação entre componentes pode adicionar latência e complexidade.

É nesse cenário que o cache emerge como um dos pilares mais importantes para a otimização. Ele atua como um atalho inteligente, armazenando cópias de dados frequentemente acessados mais perto de quem os solicita, seja o próprio usuário ou um servidor intermediário. Ao evitar que cada requisição precise percorrer todo o caminho até a origem dos dados, o cache não só acelera a entrega de conteúdo, mas também reduz a carga sobre os servidores e a largura de banda utilizada, resultando em uma experiência de usuário superior e custos operacionais mais baixos.

A Necessidade Inegável do Cache na Web Moderna

Pense na sua rotina diária: você acessa os mesmos sites, verifica as mesmas redes sociais, talvez até use os mesmos aplicativos de banco. Em cada uma dessas interações, seu dispositivo está fazendo requisições a servidores distantes, buscando dados e recursos para exibir o conteúdo que você deseja. Sem otimização, cada clique, cada rolagem de página, significaria uma nova viagem completa até o servidor de origem, um processo que consome tempo, largura de banda e recursos computacionais.

Em um mundo onde a paciência digital é escassa e a concorrência por atenção é feroz, a velocidade de carregamento de uma aplicação web pode ser o fator decisivo entre o sucesso e o fracasso. Além da experiência do usuário, a performance impacta diretamente o SEO (Search Engine Optimization), as taxas de conversão e até mesmo os custos de infraestrutura. Para empresas que operam em escala, cada milissegundo economizado em milhões de requisições se traduz em economias substanciais e em uma vantagem competitiva.

> 💡 **Analogia do Restaurante**
>
> Imagine que você tem um restaurante muito popular. Em vez de preparar cada prato do zero a cada pedido, você pode pré-preparar os ingredientes mais comuns ou até mesmo alguns pratos inteiros que sabe que sairão rapidamente. O cache faz exatamente isso!

O cache surge, então, como uma solução elegante para esse desafio. Ele armazena cópias de dados e recursos que são frequentemente solicitados, permitindo que sejam entregues quase instantaneamente, sem a necessidade de envolver o "chef" (servidor de origem) a cada vez. Essa estratégia é ainda mais vital em arquiteturas distribuídas, onde a latência de rede entre microsserviços pode ser um gargalo significativo.

O Que é Cache e Como Ele Funciona

Em sua essência, o cache é um mecanismo de armazenamento temporário de dados. Ele guarda uma cópia de um recurso (como uma imagem, um arquivo CSS, uma página HTML ou uma resposta de API) em um local de acesso mais rápido do que sua fonte original. O objetivo principal é reduzir a latência e a carga sobre o servidor de origem, servindo as requisições a partir dessa cópia armazenada sempre que possível.

01

Requisição Chega

Cliente solicita um recurso específico

02

Verificação no Cache

Sistema verifica se possui cópia armazenada

03

Cache Hit ou Miss

Se encontrado (HIT), serve imediatamente. Se não (MISS), busca na origem

04

Armazenamento

Em caso de MISS, armazena cópia para futuras requisições

Conceitos-Chave

⏱️ Time To Live (TTL)

Define por quanto tempo uma cópia em cache é considerada válida. Após esse período, ela expira e precisa ser revalidada ou recarregada da origem.

🔄 Invalidação de Cache

Processo de remover ou marcar uma cópia como obsoleta antes que seu TTL expire, geralmente porque o conteúdo original foi atualizado.

🎯 Cache Hit/Miss

Hit: recurso encontrado no cache. Miss: recurso não encontrado, necessário buscar na origem.

⚠️ **Desafio Principal:** Gerenciar a invalidação é um dos maiores desafios no design de sistemas de cache, pois dados desatualizados podem levar a experiências inconsistentes para o usuário.

Cache no Cliente: O Poder do Seu Navegador

A forma mais imediata e comum de cache que encontramos no dia a dia é o **cache do navegador**, também conhecido como cache no cliente. Quando você visita um site, seu navegador (Chrome, Firefox, Edge, etc.) não apenas exibe o conteúdo, mas também armazena localmente uma cópia de diversos recursos estáticos, como imagens, folhas de estilo CSS, arquivos JavaScript e até mesmo fontes.

Benefícios Diretos

Velocidade

Páginas carregam muito mais rápido, latência de rede praticamente eliminada

Economia de Banda

Menos dados transferidos pela rede para usuário e servidor

Redução de Carga

Servidor não processa os mesmos recursos repetidamente

Controlando o Cache do Navegador com HTTP Headers

Para que o cache do navegador funcione de forma eficaz e inteligente, os desenvolvedores precisam ter controle sobre quais recursos devem ser armazenados, por quanto tempo e sob quais condições. É aqui que entram os **HTTP Headers** de cache. Eles são como instruções que o servidor envia ao navegador, dizendo-lhe como lidar com o recurso que acabou de receber.

Cache-Control: O Header Principal

O header mais importante e amplamente utilizado para controlar o cache no cliente é o `Cache-Control`. Ele oferece uma série de diretivas que permitem um controle granular sobre o comportamento do cache.

max-age

Especifica o tempo máximo, em segundos, que um recurso pode ser considerado "fresco" (válido) pelo navegador.

```
Cache-Control: max-age=3600
```

no-cache

Instrui o navegador a sempre revalidar o recurso com o servidor antes de usá-lo (mas ainda pode armazená-lo).

```
Cache-Control: no-cache
```

no-store

Proíbe completamente o armazenamento em cache do recurso.

```
Cache-Control: no-store
```

public

Indica que o recurso pode ser armazenado por qualquer cache (incluindo proxies intermediários).

```
Cache-Control: public, max-age=86400
```

private

Restringe o armazenamento apenas ao cache do navegador do usuário final, útil para conteúdo personalizado.

```
Cache-Control: private, max-age=600
```

🎯 Ponto-Chave

Compreender e aplicar corretamente essas diretivas é essencial para balancear performance e a garantia de que o usuário sempre veja o conteúdo mais atualizado.

A Inteligência do ETag: Validação Eficiente

Mesmo com o `Cache-Control: max-age`, surge uma questão: e se o conteúdo de um recurso mudar no servidor antes que o max-age expire no navegador do usuário? O navegador continuaria servindo uma versão desatualizada. Para resolver esse problema de "conteúdo obsoleto" de forma eficiente, entra em cena o header **ETag (Entity Tag)**.

Como Funciona

O ETag funciona como uma "impressão digital" única para uma versão específica de um recurso. Quando o servidor envia um recurso, ele inclui um ETag no cabeçalho da resposta. O navegador armazena esse ETag junto com o recurso em cache.

💾 Exemplo de ETag

```
ETag: "33a64df551425fcc55e4d42a148795d9f25f89d4"
```

Uma string única que representa a versão exata do recurso

Cliente Envia

Requisição com `If-None-Match: [ETag]`

Servidor Compara

Verifica se ETag atual é igual ao recebido

Resposta Inteligente

304 Not Modified (igual) ou 200 OK (diferente)

Benefícios da Validação com ETag

- **Economia de largura de banda:** Evita o download completo do recurso quando ele não foi alterado
- **Validação precisa:** Detecta mudanças no conteúdo de forma confiável
- **Revalidação eficiente:** Torna o processo de verificação muito mais rápido
- **Melhor experiência:** Usuário sempre recebe conteúdo atualizado quando necessário

Outros Headers Importantes para Cache no Cliente

Embora `Cache-Control` e `ETag` sejam os protagonistas, outros HTTP Headers desempenham papéis importantes na gestão do cache do navegador, especialmente em cenários legados ou específicos. Conhecê-los ajuda a entender o panorama completo e a depurar comportamentos de cache.

📅 Expires

Um dos headers mais antigos. Especifica uma data e hora absolutas no futuro em que o recurso em cache deve ser considerado obsoleto.

```
Expires: Wed, 21 Oct 2025 07:28:00 GMT
```

Nota: Cache-Control: max-age é geralmente preferido por ser mais flexível e ter precedência sobre Expires.

🕐 Last-Modified / If-Modified-Since

`Last-Modified` indica a data e hora da última modificação do recurso. O navegador pode enviá-la de volta em `If-Modified-Since`.

```
Last-Modified: Tue, 15 Nov 2024 12:45:26 GMT
If-Modified-Since: Tue, 15 Nov 2024 12:45:26 GMT
```

O servidor responde com 304 Not Modified se não houver mudanças.

🔀 Vary

Crucial para garantir que o cache armazene diferentes versões de um recurso quando a resposta depende de outros headers da requisição.

```
Vary: Accept-Encoding, User-Agent
```

Indica que o servidor pode enviar diferentes versões (ex: comprimido ou não) dependendo do que o navegador aceita.

⚠️ Atenção

Embora ETag seja mais preciso (pode detectar mudanças que não alteram a data de modificação, como metadados), `Last-Modified` ainda é amplamente usado, especialmente para arquivos estáticos.

Cache no Gateway e Proxies Reversos

A Camada Intermediária

Enquanto o cache no cliente é poderoso para otimizar a experiência do usuário individual, ele não resolve o problema da carga sobre o servidor de origem quando muitos usuários acessam o mesmo conteúdo. É aqui que entra o **cache no gateway**, uma camada de cache que reside entre o cliente e o servidor de aplicação, geralmente implementada por **proxies reversos** ou **API Gateways**.

Benefícios Significativos

1 Redução da carga no servidor de origem

Muitas requisições são interceptadas e respondidas pelo gateway, aliviando o servidor de aplicação.

2 Melhora da performance para múltiplos usuários

Conteúdo popular é servido rapidamente para todos, não apenas para quem já o acessou.

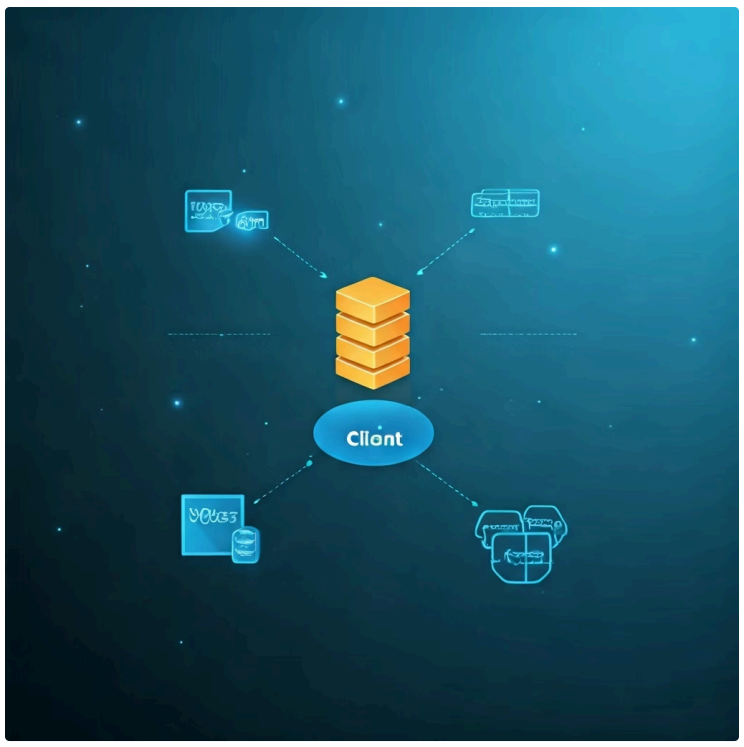
3 Resiliência

Em caso de falha do servidor de origem, o gateway pode continuar servindo conteúdo em cache.

4 Segurança e controle

Proxies reversos e API Gateways também podem adicionar camadas de segurança, balanceamento de carga e roteamento.

Varnish Cache: O Acelerador HTTP Dedicado



Quando falamos de cache no gateway com foco em alta performance para conteúdo HTTP, o **Varnish Cache** é uma das soluções mais renomadas e dedicadas. Ele não é um servidor web completo como Nginx ou Apache, mas sim um proxy reverso HTTP projetado especificamente para acelerar aplicações web, armazenando em memória (ou disco) respostas HTTP completas.

Como o Varnish Funciona

01

Posicionamento

Varnish se posiciona na frente do seu servidor web ou API Gateway

02

Interceptação

Quando uma requisição HTTP chega, o Varnish a intercepta

03

Verificação

Se a resposta estiver em cache e válida, entrega quase instantaneamente

04

Armazenamento

Se não estiver em cache, encaminha para origem, armazena e entrega

🚀 Poder da VCL

Uma das maiores forças do Varnish é sua flexibilidade, proporcionada pela **VCL (Varnish Configuration Language)**. Essa linguagem permite que os desenvolvedores escrevam regras complexas sobre como o Varnish deve lidar com as requisições: quais URLs cachear, por quanto tempo, como lidar com cookies, como invalidar o cache, e muito mais.

Alta Performance

Otimizado para servir conteúdo HTTP em alta velocidade, utilizando memória eficientemente

Flexibilidade VCL

Linguagem de configuração poderosa para regras complexas de cache

Escalabilidade

Ideal para cenários de alta demanda e otimização de conteúdo dinâmico

Configurando Varnish: Uma Visão Geral do VCL

A verdadeira magia do Varnish reside na sua capacidade de ser programado através da **VCL (Varnish Configuration Language)**. A VCL é uma linguagem de domínio específico que permite aos administradores e desenvolvedores definir políticas de cache extremamente detalhadas e personalizadas. É como escrever um roteiro para o Varnish seguir a cada requisição HTTP.

Sub-rotinas Principais do VCL

1

vcl_recv

Primeira rotina executada quando uma requisição chega. Decide se a requisição deve ser ignorada pelo cache, tratada de forma especial, ou prosseguir para busca no cache.

2

vcl_hash

Define como o Varnish gera uma chave de hash para identificar um objeto em cache. Crucial para garantir que diferentes requisições encontrem a mesma entrada.

3

vcl_fetch

Executada após o Varnish buscar o recurso do servidor de origem. Pode modificar a resposta antes de armazená-la ou decidir se deve ser cacheada.

4

vcl_deliver

Executada antes de entregar a resposta ao cliente. Pode ser usada para adicionar ou remover headers.

Exemplo Simplificado de VCL

```
# Exemplo simplificado de VCL
vcl 4.1;

backend default {
    .host = "127.0.0.1";  # Seu servidor de aplicação
    .port = "8080";
}

sub vcl_recv {
    # Remover cookies de requisições para assets estáticos para permitir cache
    if (req.url ~ "\\.(css|js|png|gif|jpg|jpeg|webp|svg|ico)$") {
        unset req.http.Cookie;
    }

    # Nunca cachear páginas de administração
    if (req.url ~ "^/admin/") {
        return (pass);  # Passa a requisição diretamente para o backend
    }
}

sub vcl_backend_response {
    # Cachear tudo por 1 hora por padrão, a menos que o backend diga o contrário
    if (beresp.ttl < 0s) {
        set beresp.ttl = 1h;
    }
}
```

💡 Flexibilidade em Ação

Com a VCL, é possível cachear todas as imagens por uma hora, mas nunca cachear páginas de login, ou até mesmo cachear diferentes versões de uma página baseadas em headers específicos do cliente. Essa flexibilidade é o que torna o Varnish uma ferramenta tão poderosa para otimização de performance em larga escala.

Nginx como Proxy Reverso e Cache

Enquanto o Varnish é um especialista em cache HTTP, o **Nginx** é um verdadeiro canivete suíço no mundo dos servidores web. Conhecido por sua alta performance e baixo consumo de recursos, o Nginx é amplamente utilizado não apenas como servidor web estático, mas também como um poderoso proxy reverso, balanceador de carga e, crucialmente para nossa discussão, como um eficiente servidor de cache.

Vantagens do Nginx para Cache

- **Versatilidade**

  Se você já utiliza o Nginx como proxy reverso, adicionar cache é uma extensão natural

- **Consolidação**

  Permite consolidar várias funções em uma única infraestrutura, simplificando a gestão

- **Performance**

  Alta eficiência no processamento de requisições e baixo consumo de recursos

- **Configuração Robusta**

  Diretivas de configuração cobrem a maioria dos cenários de cache no gateway

O Nginx pode interceptar requisições, verificar se a resposta está em seu cache local e, se estiver, servi-la diretamente. Caso contrário, ele encaminha a requisição para o servidor de aplicação, armazena a resposta e a entrega ao cliente. Embora o Nginx não ofereça a mesma flexibilidade de programação via VCL que o Varnish, suas diretivas de configuração são robustas o suficiente para cobrir a maioria dos cenários de cache no gateway, tornando-o uma escolha popular para muitas arquiteturas.

Configurando Cache no Nginx: Diretivas Essenciais

Configurar o cache no Nginx envolve algumas diretivas-chave que definem onde o cache será armazenado, por quanto tempo e sob quais condições. A beleza do Nginx está na sua sintaxe de configuração clara e concisa, que permite implementar um cache robusto com poucas linhas.

1. Definindo o Caminho do Cache: proxy_cache_path

A primeira diretiva essencial é `proxy_cache_path`. Ela define o diretório no disco onde o Nginx armazenará os arquivos cacheados, além de configurar o tamanho da zona de memória compartilhada para as chaves de cache, o tamanho máximo do cache em disco e o tempo de inatividade após o qual os itens são removidos.

```
# Exemplo de proxy_cache_path
proxy_cache_path /var/cache/nginx/my_cache
   levels=1:2
   keys_zone=my_cache:10m
   max_size=10g
   inactive=60m
   use_temp_path=off;
```

/var/cache/nginx/my_cache

Caminho para o diretório de cache no disco

levels=1:2

Cria uma estrutura de diretórios para distribuir os arquivos

keys_zone=my_cache:10m

Define uma zona de memória compartilhada de 10MB para armazenar as chaves de cache e metadados

max_size=10g

Tamanho máximo do cache em disco (10 GB)

inactive=60m

Remove itens que não foram acessados por 60 minutos

use_temp_path=off

Evita cópias temporárias, melhorando a performance

2. Habilitando e Configurando o Cache

Dentro de um bloco `location` ou `server`, você habilita o cache usando `proxy_cache` e define as regras de validade com `proxy_cache_valid`.

```
server {
   listen 80;
   server_name example.com;

   location / {
      proxy_pass http://my_backend_server;  # Seu servidor de aplicação
      proxy_cache my_cache;  # Usa a zona de cache definida acima
      proxy_cache_valid 200 302 10m;  # Cacheia respostas 200 e 302 por 10 minutos
      proxy_cache_valid 404 1m;  # Cacheia respostas 404 por 1 minuto
      proxy_cache_bypass $http_pragma $http_authorization;  # Não cacheia se houver certos headers
      add_header X-Cache-Status $upstream_cache_status;  # Adiciona header para depuração
   }
}
```

proxy_cache my_cache

Ativa o cache para esta localização, usando a zona my_cache

proxy_cache_valid

Define o TTL para diferentes códigos de status HTTP

proxy_cache_bypass

Especifica condições sob as quais o cache deve ser ignorado

add_header X-Cache-Status

Útil para depuração, mostra se a resposta veio do cache (HIT), da origem (MISS), etc.

🎯 Dica Prática

Com essas diretivas, o Nginx se torna um poderoso aliado na otimização da performance de suas aplicações. O header `X-Cache-Status` é especialmente útil durante o desenvolvimento e depuração para entender o comportamento do cache.

Estratégias de Invalidação e Coerência do Cache

> *"Existem apenas duas coisas difíceis na ciência da computação: invalidação de cache e nomear coisas."*

Essa frase é um clássico por um bom motivo. Manter o cache atualizado e coerente com a fonte de dados original é um desafio constante. Um cache desatualizado pode levar a informações incorretas sendo exibidas aos usuários, causando frustração e até mesmo problemas de negócio.

Estratégias de Invalidação

Expiração baseada em TTL

A estratégia mais simples. Definimos um tempo limite para o cache, após o qual o item é automaticamente considerado obsoleto e precisa ser revalidado ou recarregado. Funciona bem para conteúdo que muda previsivelmente ou que pode tolerar um pequeno atraso na atualização.

Purging Manual

Muitos sistemas de cache (como Varnish e Nginx, com módulos adicionais) permitem que você envie uma requisição específica para "limpar" um item ou um grupo de itens do cache. Útil quando uma atualização de conteúdo é publicada e você precisa garantir que todos os usuários vejam a nova versão imediatamente.

Cache-Busting (Versionamento de URLs)

Para recursos estáticos (CSS, JS, imagens), uma técnica comum é incluir um hash ou uma versão no nome do arquivo ou na URL (ex: `style.css?v=123` ou `style.123.css`). Quando o arquivo muda, o hash/versão muda, e a URL se torna única, forçando o download da nova versão.

Tags de Cache (Varnish ESI)

O Varnish, com sua capacidade de ESI (Edge Side Includes), pode cachear partes de uma página de forma independente. Isso permite invalidar apenas um fragmento de conteúdo, em vez da página inteira, aumentando a granularidade e a eficiência.

⚠️ Desafio em Sistemas Distribuídos

A escolha da estratégia depende da criticidade do conteúdo, da frequência de atualização e da tolerância a dados desatualizados. Em sistemas distribuídos, a coordenação da invalidação de cache entre múltiplos serviços e camadas de cache é um desafio complexo que exige design cuidadoso.

Cache no Cliente vs. Cache no Gateway: Quando Usar Cada Um

Compreendemos agora as duas principais camadas de cache abordadas nesta aula: o cache no cliente (navegador) e o cache no gateway (proxies reversos como Varnish e Nginx). Ambas são ferramentas poderosas, mas servem a propósitos ligeiramente diferentes e se complementam para formar uma estratégia de cache robusta.

🖥️ Cache no Cliente

A camada mais próxima do usuário.

→ Recursos estáticos

Imagens, CSS, JavaScript, fontes que raramente mudam

→ Conteúdo personalizado

Dados específicos de um usuário (com `private`)

→ Redução de latência individual

Acelera o carregamento para o usuário que já acessou

→ Economia de largura de banda

Reduz o consumo de dados do cliente

🌐 Cache no Gateway

Ponto de cache compartilhado para múltiplos usuários.

→ Conteúdo dinâmico popular

Respostas de API, páginas HTML iguais para muitos usuários

→ Redução da carga no servidor

Alivia a pressão sobre servidores de aplicação e banco de dados

→ Melhora da performance geral

Acelera o acesso para todos os usuários, mesmo na primeira visita

→ Resiliência

Pode servir conteúdo em cache mesmo se o servidor estiver indisponível

---

Comparação Direta

Conceito	Âmbito/Aplicação	Base/Origem	Exemplo
Cache Cliente	Usuário final, dispositivo local	HTTP Headers (Cache-Control, ETag)	Imagens de perfil, arquivos CSS de um site
Cache Gateway	Servidor intermediário, rede	Varnish, Nginx, CDN	Respostas de API de produtos, página inicial de um e-commerce

🎯 Estratégia Ideal

A melhor estratégia é, geralmente, uma **combinação de ambos**. O cache no gateway pode servir conteúdo rapidamente para a primeira visita de um usuário, enquanto o cache no cliente otimiza as visitas subsequentes. Juntos, eles criam uma hierarquia de cache que maximiza a performance e a escalabilidade, garantindo que as requisições sejam atendidas pela camada de cache mais próxima e eficiente possível.

Consolidação e Próximos Passos

Nesta primeira parte sobre estratégias de cache, exploramos as camadas mais próximas ao usuário: o cache no cliente e o cache no gateway. Vimos como o navegador, utilizando HTTP Headers como `Cache-Control` e `ETag`, pode armazenar recursos localmente para acelerar carregamentos subsequentes e economizar largura de banda. Em seguida, mergulhamos no mundo do cache no gateway, entendendo como proxies reversos como Varnish e Nginx atuam como intermediários inteligentes, aliviando a carga sobre os servidores de origem e melhorando a performance para múltiplos usuários.

✅ Cache no Cliente

HTTP Headers, navegadores, recursos estáticos, performance individual

✅ Cache no Gateway

Varnish, Nginx, proxies reversos, conteúdo compartilhado, redução de carga

✅ Estratégias de Invalidação

TTL, purging, cache-busting, coerência de dados

Em Prática: Checklist de Implementação

- ✓ Sempre defina `Cache-Control` e `ETag` para seus recursos estáticos e, quando apropriado, para respostas de API
- ✓ Considere usar um proxy reverso como Nginx ou Varnish na frente de suas aplicações para cachear conteúdo dinâmico e aliviar seus servidores
- ✓ Planeje suas estratégias de invalidação de cache cuidadosamente para evitar conteúdo obsoleto
- ✓ Monitore o status do cache (HIT/MISS) para entender a eficácia de suas configurações

📚 Autoavaliação

1

Questão 1

Qual HTTP Header é primariamente responsável por indicar ao navegador por quanto tempo um recurso pode ser considerado válido antes de precisar ser revalidado?

a) Expires
b) ETag
c) Cache-Control
d) Last-Modified

2

Questão 2

Qual o principal benefício do uso do ETag em comparação com apenas Cache-Control: max-age?

a) Reduz o tempo de vida do cache
b) Permite que o servidor saiba se o recurso foi modificado sem precisar enviar o conteúdo completo
c) Força o navegador a sempre baixar a versão mais recente
d) É usado exclusivamente para cache de imagens

3

Questão 3

Em um cenário onde múltiplos usuários acessam frequentemente o mesmo conteúdo dinâmico de uma API, qual tipo de cache seria mais eficaz?

a) Cache de navegador (Cache-Control: private)
b) Cache de navegador (Cache-Control: no-store)
c) Cache no gateway (ex: Varnish ou Nginx)
d) Cache de banco de dados

4

Questão 4

Qual das seguintes diretivas do Nginx é utilizada para definir o caminho no disco onde os arquivos cacheados serão armazenados?

a) proxy_cache
b) proxy_cache_valid
c) proxy_cache_path
d) proxy_pass

5

Questão 5

Explique a importância da estratégia de "cache-busting" para recursos estáticos e como ela se relaciona com a invalidação de cache.

Gabarito e Recursos Adicionais

✅ Gabarito das Questões

Questão 1

c) Cache-Control

Questão 2

b) Permite que o servidor saiba se o recurso foi modificado sem precisar enviar o conteúdo completo

Questão 3

c) Cache no gateway (ex: Varnish ou Nginx)

Questão 4

c) proxy_cache_path

📖 Recursos Adicionais

MDN Web Docs - HTTP Caching

Documentação abrangente sobre headers de cache HTTP e melhores práticas de implementação

Varnish Cache Documentation

Guias detalhados sobre VCL e configuração do Varnish para cenários avançados

Nginx Caching Guide

Artigos sobre como configurar o Nginx para cache de proxy reverso e otimização

---

🚀 Próxima Aula

Aula 27 – Estratégias de Cache – Parte 2: Cache Distribuído

Exploraremos as complexidades e os benefícios do cache em ambientes distribuídos, como Redis e Memcached, e como eles se integram em arquiteturas de microsserviços e serverless.

NOTA IMPORTANTE: As informações regulatórias/legais/técnicas desta aula estão atualizadas até 2025. Consulte sempre fontes oficiais para verificar alterações.