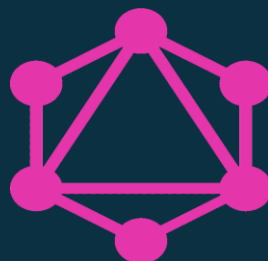


# GraphQL & Elasticsearch

## *O Caminho do Ninja das APIs*



Domine consultas precisas e buscas ultra-rápidas com furtividade e poder.



# ELASTICSEARCH

O Elasticsearch é um mecanismo de busca e análise distribuído desenvolvido com base no Apache Lucene. Ele foi projetado para busca de texto completo de alto desempenho, filtragem e análise de dados em grandes volumes de dados estruturados e não estruturados.



## Principais Recursos do Elasticsearch

- Busca e análises quase em tempo real.
- Arquitetura distribuída – escala horizontalmente.
- Busca de texto completo com pontuação de relevância, stemming, sinônimos, etc.
- Consulta flexível usando JSON DSL.
- Armazena dados estruturados e não estruturados (baseados em JSON).
- Suporta agregações (como SQL GROUP BY + métricas).
- Alta disponibilidade via replicação.

# GRAPHQL

O GraphQL é uma linguagem de consulta para APIs e também um ambiente de execução para essas consultas. Ele foi desenvolvido pelo Facebook e permite que o cliente (frontend ou consumidor da API) peça exatamente os dados que precisa, em uma única requisição — nem mais, nem menos.



- Pontos importantes do GraphQL  
Um único endpoint (Em vez de múltiplas rotas como no REST)
- Consultas personalizadas
- Esquema fortemente tipado (modelo de dados bem definido)
- Introspecção embutida (facilita a geração de documentação)

# União de GraphQL e Elasticsearch

Unir GraphQL (linguagem de consulta para APIs) com Elasticsearch (motor de busca e análise de dados) oferece uma camada de acesso inteligente + motor de busca veloz, formando uma arquitetura robusta para sistemas que exigem performance, precisão e flexibilidade. Isso porque:

## GraphQL

Interface inteligente para dados

Permite consultar apenas os campos desejados

Ótimo para manipular dados relacionais (ex: usuários, pedidos)

Suporta mutations e subscriptions (tempo real)

Ideal para APIs de frontend (React, Vue, mobile)

## Elasticsearch

Busca full-text extremamente rápida

Suporta filtros, ordenações, sugestões, autocompletar

Ótimo para indexar e consultar documentos densos (ex: artigos, logs)

Suporta consultas agregadas e análises complexas

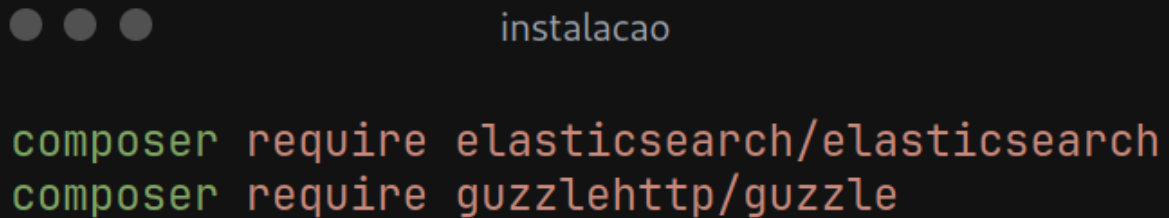
Ideal para acelerar buscas e aliviar o banco de dados principal

## Como funciona?

- Problema: O servidor GraphQL está sobrecarregado com as mesmas consultas frequentes.
- Solução: Salvar os resultados no Elasticsearch, usando a query como chave (cacheKey), e servir do cache sempre que possível.
- Resultado: Menos carga no backend e resposta mais rápida ao usuário final.

## Exemplo de implementação - PHP

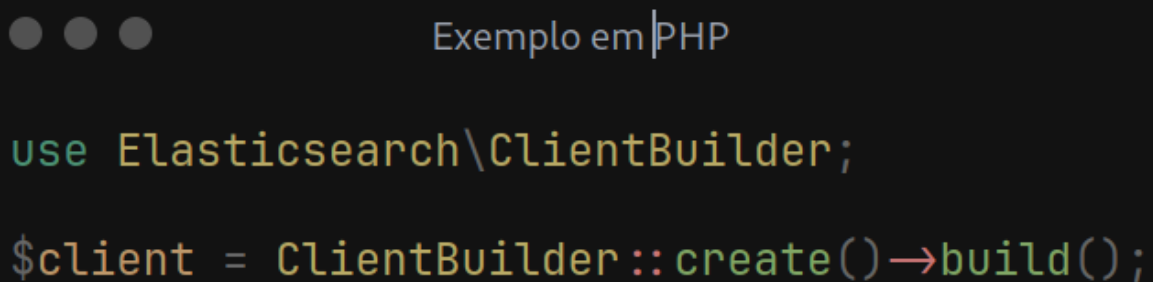
Da mesma forma como ocorre com outras stacks / linguagens, é necessário instalar as bibliotecas:



```
composer require elasticsearch/elasticsearch
composer require guzzlehttp/guzzle
```

- Conectar ao ElasticSearch

Use um hash da query GraphQL com variáveis para identificar os requests únicos.



```
use Elasticsearch\ClientBuilder;

$client = ClientBuilder::create()→build();
```

# Gerar uma Chave de Pesquisa

Criar uma chave de pesquisa com um hash GraphQL e variáveis para identificar os requests únicos.

```
Verificar Dados no Caching com Elasticsearch|  
  
function generateCacheKey(string $query, array $variables = []): string {  
    return md5($query . json_encode($variables));  
}
```

## Buscar com a API GraphQL

Seja em uma dupla de dados ou em um encapsulamento mais complexo, a busca de informações pela API ocorre no seguinte fluxo

```
Buscar sua API GraphQL  
  
use GuzzleHttp\Client;  
  
$gqlClient = new Client();  
  
$response = $gqlClient->post('https://sua-graphql-api.com.br/graphql', [  
    'json' => [  
        'query' => $query,  
        'variables' => $variables  
    ]  
]);  
  
$body = json_decode($response->getBody(), true);  
$data = $body['data'] ?? null;
```



# Aplicar Caching na resposta recebida

Encapsular os dados recebidos no caching é a chave para a coexistência entre os 2 sistemas

```
Aplicar Caching na resposta

$client->index([
    'index' => 'graphql_cache',
    'id'     => $cacheKey,
    'body'   => [
        'data' => $data,
        'cached_at' => date('c')
    ]
]);
```

## Opcional: Configurar a expiração / TTL

O Elasticsearch não tem mais o TTL nativamente, logo, gerencie o Index do ciclo de vida, ou delete manualmente registros no cache antigo:

```
Deletar manualmente cache antigo

$client->delete([
    'index' => 'graphql_cache',
    'id'     => $oldCacheKey
]);
```

# Tratar todo o conjunto em Função

O uso de uma função auxiliar para organizar as diferentes partes do código e estabelecendo um gatilho para o acionamento deste contexto

```
Incluir tudo em uma função

function graphqlWithCache($query, $variables = []) {
    $client = ClientBuilder::create()→build();
    $gqlClient = new Client();

    $cacheKey = generateCacheKey($query, $variables);

    // Check cache
    try {
        $cached = $client→get(['index' ⇒ 'graphql_cache', 'id' ⇒ $cacheKey]);
        return $cached['_source']['data'];
    } catch (\Exception $e) {
        // Cache miss or error
    }

    // Fetch GraphQL
    $response = $gqlClient→post('https://sua-graphql-api.com/graphql', [
        'json' ⇒ ['query' ⇒ $query, 'variables' ⇒ $variables]
    ]);
    $data = json_decode($response→getBody(), true)['data'] ?? null;

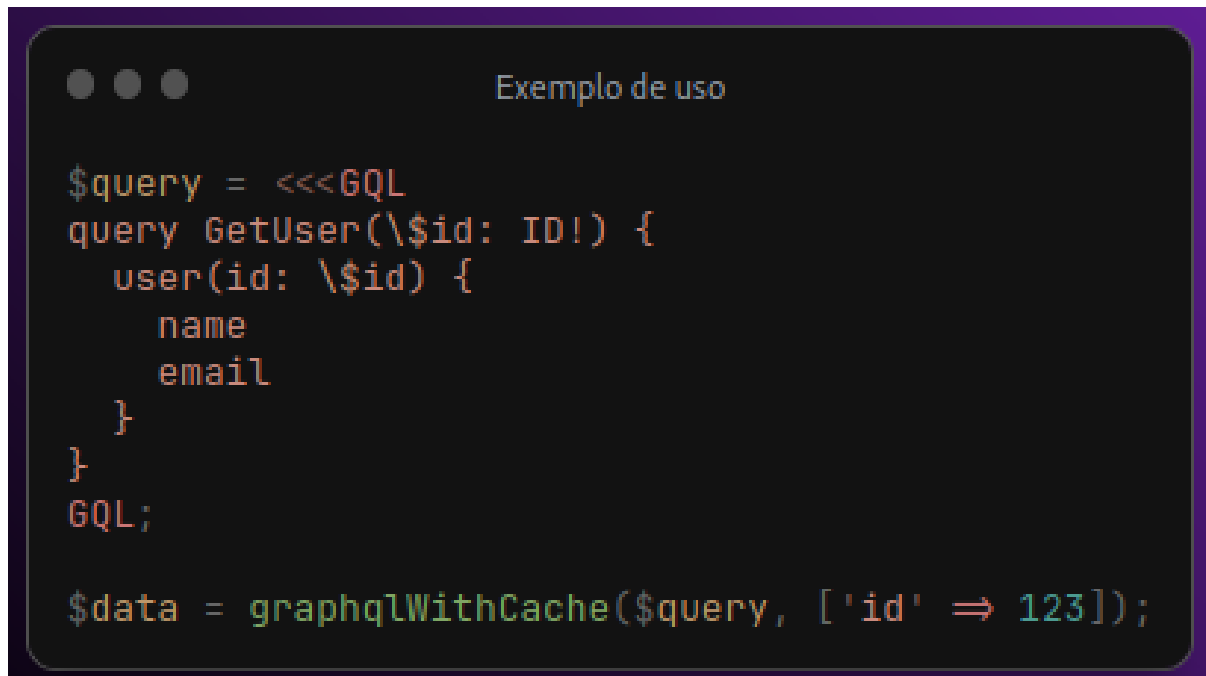
    // Store in cache
    $client→index([
        'index' ⇒ 'graphql_cache',
        'id' ⇒ $cacheKey,
        'body' ⇒ ['data' ⇒ $data, 'cached_at' ⇒ date('c')]
    ]);

    return $data;
}
```



## Exemplo de uso

Como chamar a função da maneira correta passando os parâmetros necessários a fim de acionar o gatilho para o caching chamando o `graphqlWithCache`

A screenshot of a code editor window with a dark background and a purple border. The window title is "Exemplo de uso". It contains two lines of code. The first line defines a GraphQL query named "getUser" that takes an "id" parameter and returns the "name" and "email" of the user. The second line uses the "graphqlWithCache" function, passing the query and a cache key that maps "id" to the value "123".

```
Exemplo de uso

$query = <<<GQL
query GetUser(\$id: ID!) {
  user(id: \$id) {
    name
    email
  }
}
GQL;

$data = graphqlWithCache($query, ['id' => 123]);
```

Este foi apenas um artigo breve, direto ao ponto, sobre como orquestrar as consultas do GraphQL com o caching do Elasticsearch