Usando Elasticsearch como banco de dados NOSQL

O Elasticsearch pode armazenar documentos JSON e assim agir como um banco de dados NOSQL.

Diferente do mongodo e outros bancos de dados, o elasticsearch cria índices para todos os campos do documento JSON e se você não quiser um índice precisa indicar na criação do mapping do index.

No exemplo vamos usar o caso de uso de armazenar um usuário com nome, data de registro, idade e endereço. Para não gerar o índice de todos os campos, não vamos indexar os campos do endereço.

Exemplo de mapeamento:

```
{
  "mappings": {
   "properties": {
    "name": { "type": "text" },
     "age": { "type": "integer" },
     "date_of_registration": { "type": "date" },
     "address": {
      "type": "object",
      "properties": {
       "city": { "type": "text", "index": false },
       "state": { "type": "text", "index": false },
       "street": { "type": "text", "index": false },
       "number": { "type": "integer", "index": false }
      }
    }
   }
 }
}
```

Passando index=false no mapeamento indicamos ao elasticsearch que não queremos criar indices para esse campo

Para criar esse indice basta realizar um PUT em https://enderecoelastic/nome_do_index com o mapeamento como corpo.

Todo o gerenciamento de index, operações de insert, update, delete e search são realizados por API rest no elasticsearch.

```
curl --location --request PUT 'https://localhost:9200/users' \
--header 'Content-Type: application/json' \
--header 'Authorization: Basic ZWxhc3RpYzpHK1JlaGQwMGFpQmc4S3BQS0hOZg==' \
--data '{
    "mappings": {
        "properties": {
            "name": { "type": "text" },
            "age": { "type": "integer" },
            "date_of_registration": { "type": "date" },
            "address": {
```

```
"type": "object",

"properties": {

    "city": { "type": "text", "index": false },

    "state": { "type": "text", "index": false },

    "street": { "type": "text", "index": false },

    "number": { "type": "integer", "index": false }

    }

}

}
```

A criação de novos campos é uma tarefa simples, pois podemos alterar o mapeamento para adicionar um novo campo mas por padrão o elasticsearch aceita campos dinâmicos, portanto se mandarmos um json com um campo novo o elasticsearch vai estimar o tipo do campo e criar o indice apropriado.

Para campos dinâmicos o elasticsearch sempre cria indices.

Ex:

Se criarmos um usuário com o seguinte payload:

```
"name": "João Silva",
"age": 30,
"nrc":11987,
"address": {
  "city": "São Paulo",
  "state": "SP",
  "street": "Rua Principal",
  "number": 100
}
```

O campo nrc seria criado no mapeamento do index User automaticamente e será criado um índice para ele. Esse é o comportamento padrão do elasticsearch. Podemos alterar esse comportamento na criação do index para proibir a criação de campos dinâmicos por exemplo.

Para mais informações, segue link da documentação:

https://www.elastic.co/quide/en/elasticsearch/reference/current/dynamic-field-mapping.html

Se por algum motivo for necessário indexar um campo que não tem indice será preciso:

- Criar um novo index com o mapeamento atualizado.
- Reindexar os dados do index antigo para o novo index.
- Apagar o index antigo (opcional).
- Renomear o novo index para o nome original, se necessário.

Por esse motivo o caso de uso mais indicado para o elasticsearch é o de armazenamento de logs (como logs de requisição) onde as ferramentas de busca textual serão um diferencial.

Implementação

Como detalhes de implementação, como tanto o mongodo e o elasticsearch trabalho com documentos json seu uso no desenvolvimento.

Por exemplo para a struct abaixo:

A implementação abaixo salva um usuario na base do elastic em go:

```
func (ur *UserElasticRepository) CreateUser(user domain.User) error {
   user.DateOfRegistration = time.Now()

_, err := ur.client.Index().
        Index("users").
        BodyJson(user).
        Do(context.Background())

if err != nil {
        log.Printf("Error saving user to Elasticsearch: %v", err)
        return err
   }

   return nil
}
```

A mesma implementação em Mongo:

```
func (ur *UserMongoRepository) CreateUser(user domain.User) error {
   user.DateOfRegistration = time.Now()
   fmt.Printf("User %v\n", user)
```

```
__, err :=
ur.client.Database("test").Collection("users").InsertOne(context.Backgr
ound(), user)
    if err != nil {
        log.Printf("Error saving user to MongoDB: %v", err)
        return err
    }
    return nil
}
```

Para a busca por um campo (no caso o nrc)

Elastic:

```
func (ur *UserElasticRepository) FindUserByNRC(nrc int) (*domain.User,
error) {
        Index("users").
       Query(elastic.NewTermQuery("nrc", nrc)).
       Do(context.Background())
   if err != nil {
       log.Printf("Error fetching user from Elasticsearch: %v", err)
       return nil, err
   if searchResult.Hits.TotalHits.Value == 0 {
searchResult.Each(reflect.TypeOf(domain.User{})) {
       user = item.(domain.User)
       user.ID = searchResult.Hits.Hits[idx].Id
   return &user, nil
```

No mongo:

```
func (ur *UserMongoRepository) FindUserByNRC(nrc int) (*domain.User,
error) {
```

```
var user domain.User
    err :=
ur.client.Database("test").Collection("users").FindOne(context.Backgrou
nd(), bson.M{"nrc": nrc}).Decode(&user)
    if err != nil {
        if err == mongo.ErrNoDocuments {
            return nil, nil
        }
        log.Printf("Error fetching user from MongoDB: %v", err)
        return nil, err
    }
    return &user, nil
}
```

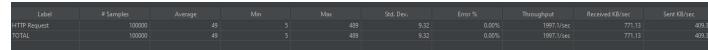
Portanto, em termos de desenvolvimento é possível criar uma camada de repositório para separar o acesso ao banco e isolar o mongo ou o elasticsearch.

Teste de Performance

Foram realizados testes de stress para o cadastro de 100.000 usuários aleatórios gerados por 100 threads (users) rodando ao mesmo tempo tanto no mongo quanto no elastic.

No mongo notamos que o Throughput da aplicação é bem maior.

Resultado Elastic:

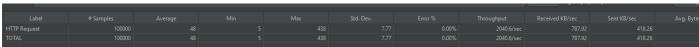


Resultado Mongo:



Também foram realizados testes de update, onde é realizado uma busca por um registro e o update dos campos por valores aleatórios.

Resultado Elastic:



Resultado Mongo:

Label					Throughput			
HTTP Request								
TOTAL			8.52	0.00%	6095.0/sec	1762.79	1249.30	

Considerações Finais

O melhor caso de uso para o elasticsearch seria a ingestão de log onde as ferramentas de busca textual irão ajudar a encontrar os erros

Para o armazenamento de dados transacionais em um banco de dados NOSQL o mongodo segue como melhor opção devido ao Throughput maior e mais flexibilidade no uso de índices, visto que o esquema desses dados sofrem mais alterações ao longo do ciclo de vida da aplicação e frequentemente é necessário criar índices para campos que antes não possuía.