6 - Indexing Operations

Tuesday, 4 March, 2025 9:46 Manhã

- Configurar index a baixo nível é importante
- Tem três configurações básicas:
  - Settings
    - E.g. número de shards e réplicas
  - Mapping
    - Schema
  - Alias
    - Renomear para melhorar querying e reindexação
- Index devem ser montados com **templates**, não manualmente.
- Políticas de rollover:
  - Alterações conforme o tempo e o tamanho crescem
  - Aposentar dados
- Várias operações com índices:
  - Creating, closing, shrinking, cloning, freezing, deleting, etc

CREATING INDEXES

- Index pode ser criado implicitamente (automaticamente)
  - Basta indexar o documento onde ele deve ficar
  - Podemos alterar depois mas nem tudo é alterável num index existente
  - Para desabilitar auto create index:
    - **Listing 6.2 Disabling automatic creation of indexes**

```
PUT _cluster/settings
{
  "persistent": {
    "action.auto_create_index":false
  }
}
```

Updates the settings across the whole cluster

The changes can be persistent or transient.

Shuts down auto-creation

    - Desativar criação automática pode causar problemas por exemplo com o Kibana, que cria index ocultos para administração
    - Index oculto tem um ponto antes (e.g. .user_profiles, .admin etc)
    - Para desativar auto_create seletivamente, por padrão de nome:

    - `action.auto_create_index: [".admin*, cars*, "*books*", "-x*","-y*","+z*"]`
    - Premite index automaticos com admin e cars e books no nome, mas não permite começando com x ou y.
    - Index fora de qualquer padrão também é desabilitado
- **Mapping**:
  - CAP 04
- **Settings**
  - **Listing 6.3 Creating an index with custom settings**

```
PUT cars_index_with_sample_settings
{
  "settings": {
    "number_of_replicas": 3,
    "codec": "best_compression"
  }
}
```

Creates the index

Applies the settings

  - Dynamic setting:
    - Pode ser definito em index operante
  - Static setting:
    - Só pode ser definido em index não em operação, não depois
    - Para alterar, precisamos fechar o index e reaplicar os settings, ou recriar o index
    - Dá exception se tentar trocar

- `GET cars_index_with_sample_settings/_settings`

- **Alias**
  - Nomes alternativos
  - Pode apontar para um ou vários index
  - **Listing 6.4 Creating an index with an alias**

```
PUT cars_index_with_sample_alias
{
  "aliases": {
    "alias_for_cars_index_with_sample_alias": {}
  }
}
```

Creates the index. Declares the aliases object to configure the alias. The alias itself.

- **Criar Index Explicitamente**
  - Ideal para produção
  - "PUT cars" cria um index cars com configurações default
  - **Listing 6.5 Creating an index with custom settings**

```
PUT cars_with_custom_settings
{
  "settings":{
    "number_of_shards":3,
    "number_of_replicas":5,
    "codec": "best_compression",
    "max_script_fields":128,
    "refresh_interval": "60s"
  }
}
```

Creates an index with custom settings. The settings object encloses the required properties. Sets the number of shards to 3. Sets the number of replicas to 5. Changes the compression from its default value. Increases the maximum number of script fields from its default of 32. Changes the refresh interval from its default of 1 second.

- Para reconfigurar os settings de um index:
  a. Fechar o index atual (desabilita read/write)
  b. Criar o index novo com os novos settings
  c. Migrar os dados do index velho para o novo (reindexing)
  d. Reapontar o alias para o novo index (se houver alias)

- **Listing 6.8 Fetching the settings of multiple indexes at once**

```
GET cars1,cars2,cars3/_settings
GET cars*/_settings
```

Fetches multiple index settings. Fetches the settings for the index identified with a wildcard (*)

- **Listing 6.9 Fetching a single attribute**

```
GET cars_with_custom_settings/_settings/index.number_of_shards
```

- Para criar index com **Alias**:
  - Podemos usar API _alias ou passar no corpo da requisição
  - **Listing 6.12 Creating an alias using an `aliases` object**

```
PUT cars_for_aliases
{
  "aliases": {
    "my_new_cars_alias": {}
  }
}
```

Creates an index with an alias. Points the alias to the index.

Listing 6.13 Creating an alias using an `_alias` endpoint

Listing 6.13 Creating an alias using an `_alias` endpoint

```
PUT cars_for_aliases/_alias/my_cars_alias
```

- Podemos criar vários índices juntos apontando a o mesmo alias
- Wildcard pode ser usado aqui também, em vez da vírgula

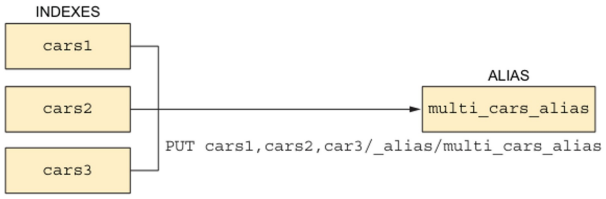

Figure 6.3 Creating an alias pointing to multiple indexes

- Neste caso com vários, pelo menos um deles deve ter o atributo *is_write_index*

```
PUT cars3
{
  "aliases": {
    "cars_alias": {
    "is_write_index":true
    }
  }
}
```

- **Migração sem Downtime**
  - Por exemplo, atualizar um index vintage_cars
    i. Criar um aliás *vintage_car_alias* apontando para *vintage_cars*
    ii. Criar um novo index *vintage_cars_new* com os novos settings desejados
    iii. Copiar (reindexar) dados do index antigo para o index novo
    iv. Recriar o alias existente para apontar para o novo index
    v. Fazer as queries serem executadas para o alias, não o index direto
    vi. Apagar o index antigo e o index novo estiver em operação
- Podemos configurar vários aliases com a API *_aliases*

Listing 6.19 Performing multiple aliasing operations

```
POST _aliases                          <- Uses the _aliases API to execute multiple actions
{
  "actions": [                         <- Lists the individual actions
    {
      "remove": {                      <- Removes the alias that points to an old index
        "index": "vintage_cars",
        "alias": "vintage_cars_alias"
      }
    },
    {
      "add": {                         <- Adds an alias that points to a new index
        "index": "vintage_cars_new",
        "alias": "vintage_cars_alias"
      }
    }
  ]
}
```

- "index" pode ser um array (daí a key vira "indices")

READING INDEXES

- Estamos lidando com index públicos

Listing 6.24 Getting individual configurations for an index

```
GET cars/_settings                     <- Gets the settings from the _settings endpoint
GET cars/_mapping                      <- Gets the mappings from
GET cars/_alias
```

Listing 6.24 Getting individual configurations for an index

```
GET cars/_settings
GET cars/_mapping
GET cars/_alias
```

Gets the settings from the _settings endpoint

Gets the mappings from the _mapping endpoint

Gets the aliases from the _alias endpoint

- Com comando "HEAD cars" nos diz se index existe
- Há também os hidden index
  - Prefixo por ".", e.g. ".admin"
  - Hoje podemos criar hidden index livremente, mas no futuro deve se tornar apenas para index de sistema
  - *GET _all* ou GET * inclui hidden index

DELETING INDEXES

- Comando "DELETE cars, movie, order"
- Também podemos usar wildcard para deletar
  - Mas para deleter também os hidden indexes, precisamos usar o _all
  - Podemos desabilitar ou não delete por wildcard/_all:
    - Default é true
    -
```
PUT _cluster/settings
{
 "transient": {
   "action.destructive_requires_name":false
 }
}
```
- Também podemos deletar apenas alias:

Listing 6.26 Deleting an alias explicitly

  - `DELETE cars/_alias/cars_alias` ← Deletes cars_alias
- Não tem volta

CLOSING AND OPENING INDEXES

- **Closing:**
  - Index fechado fica sem ler/escrever dados
  - **Listing 6.27 Closing the `cars` index indefinitely**

    `POST cars/_close`
  - Também podemos usar wildcard ou _all
    - (também requer propriedade de destructive_requires_name = True)
  - Fechar index limpa memória e disponibiliza recursos, busca requer menos recursos
  - Podemos desabilitar fechamento completamente:
    - Default é true
    - **Listing 6.30 Disabling the closing indexes feature**

```
PUT _cluster/settings
{
  "persistent": {
    "cluster.indices.close.enable":false
  }
}
```
- **Opening:**
  - Podemos simplesmente abrir um index fechado para restaurá-lo:

- **Listing 6.31 Putting the index back into operation**

```
POST cars/_open
```

  - Análogo ao close

INDEX TEMPLATE

- Pré-settar configurações (settings, mappings, alias) de um index seguindo um default definido
- Por exemplo, cada ambiente ter um template com um número de réplicas
- Se o index tiver um nome que bate com um certo padrão de nome (glob) definido, é aplicado o template ao index.
- Existem dois tipos de template:
  - Component
  - Composable
    - Composto de zero ou mais index templates components
  - * Template também pode existir em avulso.

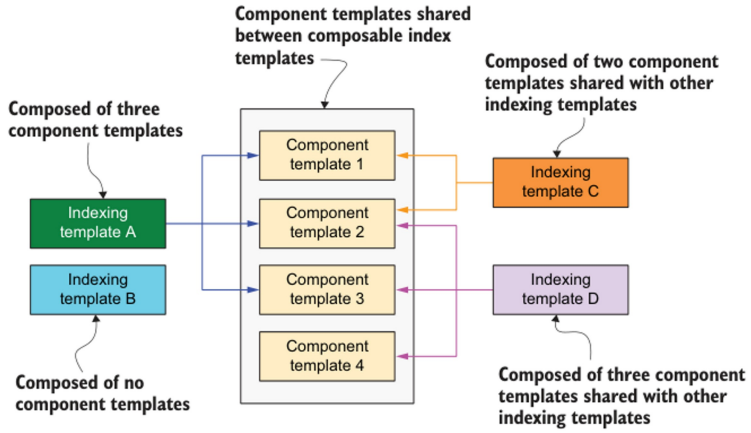

Figure 6.6 Composable (index) templates are composed of component templates.

- Assim vamos montando templates de configurações a partir de pequenos padrões menores.
  - Configurações mandadas explicitamente por comando têm prioridade sobre templates.
  - Legacy template tem prioridade menor que composable template
  - Podemos definir prioridade de template explicitamente com um número inteiro:
    - Default priority = 1

```
POST _index_template/cars_template_mar21
{
  "index_patterns": ["*cars*"],
  "priority": 20,        <-- Lower-priority index template
  "template": { ... }
}
POST _index_template/cars_template_feb21
{
  "index_patterns": ["*cars*"],

  "priority": 30,        <-- Matching template, but with a higher priority
  "template": { ... }
}
```

    - Feb21 neste exemplo tem prioridade sobre mar21
    - Aplica todas as configurações de todas os templates, mas se houver alguma igual em ambos os templates, prevalece a do template de maior prioridade

  - Se um index der match em mais de um template, usa-se a prioridade

- API _index_template para criar template:

Listing 6.32 Creating an index template

```
PUT _index_template/cars_template
{
  "index_patterns": ["*cars*"],
  "priority": 1,
  "template": {
    "mappings": {
      "properties":{
        "created_at":{
          "type":"date"
        },
        "created_by":{
          "type":"text"
        }
      }
    }
  }
}
```

- Qualquer index com "cars" no nome receberá este template

- Component template é um bloco de configurações reutilizável
  - *_component_template* endpoint

Listing 6.33 Developing a component template

```
POST _component_template/dev_settings_component_template
{
  "template":{
    "settings":{
      "number_of_shards":3,
      "number_of_replicas":3
    }
  }
}
```

  - Fará parte de um template, não é um template em si
    - Por exemplo, não tem um index pattern associado
  - Para utilizar o componente em um template:

Listing 6.35 Index template composed of component templates

```
POST _index_template/composed_cars_template
{
  "index_patterns": ["*cars*"],
  "priority": 200,
  "composed_of": ["dev_settings_component_template",
                  "dev_mapping_component_template"]
}
```

MONITORING AND MANAGING INDEXES

- Estatísticas dos indexes
  - Número total de documentos, documentos deletados, etc
  - *_stats* API

Listing 6.36 Fetching the statistics of an index

```
GET cars/_stats
```

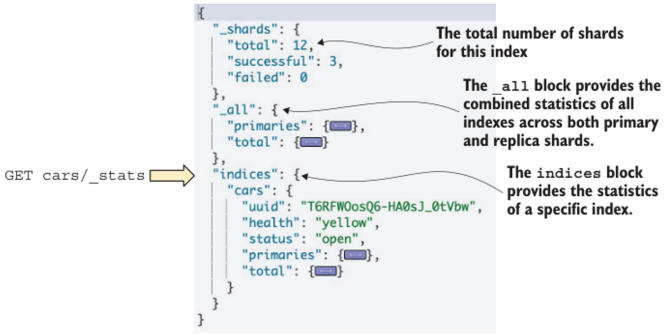

Figure 6.7 Statistics for a `cars` index

Table 6.1 Index statistics fetched by a call to the `_stats` endpoint (the list is shortened for brevity)

Statistic	Description
docs	Number of documents in the index and number of documents deleted
store	Size of the index (in bytes)
get	Number of GET operations on the index
search	Search operations including query, scroll, and suggest times.
refresh	Number of refresh operations

- Também dá pra listar vários indexes pra ter statisticas de vários index, ou então o wildcard
- Podemos querer só alguns tipos de estatísticas, como segments:

Listing 6.40 Segment statistics

```
GET cars/_stats/segments
```

This command returns the following data:

```
"segments" : {
        "count" : 1,
        "memory_in_bytes" : 1564,
        "terms_memory_in_bytes" : 736,
        "stored_fields_memory_in_bytes" : 488,
        "term_vectors_memory_in_bytes" : 0,
        "norms_memory_in_bytes" : 64,
        "points_memory_in_bytes" : 0,
        "doc_values_memory_in_bytes" : 276,
        "index_writer_memory_in_bytes" : 0,
        "version_map_memory_in_bytes" : 0,
        "fixed_bit_set_memory_in_bytes" : 0,
        "max_unsafe_auto_id_timestamp" : -1,
        "file_sizes" : { }
    }
```

- Também podemos chamar o endpoint *_segments* diretamente

ADVANCED OPERATIONS

- Repartir um index, rolling periódico etc

- **Splitting**
  - Bom quando um index tem dados demais e não queremos sobrecarregá-lo

- Dividir um index para um index com mais shards
- É simplesmente um novo index com settings diferentes e os dados copiados
- API *_split*
  - Primeiro passo é bloquear as operações de write:

Listing 6.41 Making sure the index is read-only

```
PUT all_cars/_settings                          <- Uses the _settings API
{
  "settings":{
    "index.blocks.write":"true"                 <- Closes the index for writing operations
  }
}
```

  - Daí então fazemos o split:

Listing 6.42 Splitting the `all_cars` index

```
POST all_cars/_split/all_cars_new               <- _split expects a target index.
{
  "settings": {
    "index.number_of_shards": 12                <- Sets the number of shards on the new index
  }
}
```

    - Processo síncrono (só termina quando resposta é enviada)
      - Novo index terá todos os dados do index antigo
  - **Condições:**
    - **Index target não pode já existir**
    - **Número de shards do target tem que ser um múltiplo do valor original**
    - **Target não pode ter menos shards do que o original**
    - **Node do target tem que ter espaço o suficiente**
  - Splitting só muda o número de shards, o resto, como os settings, fica igual

- **Shrinking**
  - Operação reversa do splitting: diminuir o número de shards.
  - Otimizar recursos
    - Primeiro fazemos como no split de bloquear write:
      - Também restringimos os shards existentes a um único node

Listing 6.43 Prerequisites before shrinking the index

```
PUT all_cars/_settings
{
  "settings": {
    "index.blocks.write": true,
    "index.routing.allocation.require._name": "node1"
  }
}
```

    - Daí sim fazemos o shrink:

Listing 6.44 Shrinking the index

```
PUT all_cars/_shrink/all_cars_new               <- Shrinks the source index
{
  "settings":{
    "index.blocks.write":null,                  <- Removes the read-only instruction
    "index.routing.allocation.require._name":null,   <- Sets the node name to null
    "index.number_of_shards":1,                 <- Reduces the number of shards
    "index.number_of_replicas":5
  }
}
```

    - Número de shards deve ser um fator do número de shards original
  - **Condições:**

- **Source index tem que ser desligado (tornar read-only)**
  - É uma boa ideia também desabilitar réplicas (igual a 0)
- **Target index não pode já existir**
- **Todos os shards do source index devem estar no mesmo node**
- **Número de shards do target deve ser um fator do número original**
- **O node do target index deve ter memória o suficiente**

- **Rolling Over de um alias**
  - Rolamento automático do Index atual a um index novo
  - Dados do index antigo não são copiados para o novo.
  - Antigo se torna read-only e todos os documentos novos vão para o index novo
  - Uso comum é em séries temporais (i.e. dado relativo a um período específico)
  - Roll over são basicamente dois passos:
    i. *Alias* é criado apontando ao index original (que tem que ser writeable)
    ii. ES chama a API _rollover que cria um novo index e aponta o *alias* para ele
      - Sempre tem que ter ao menos um index writeable
      - ES automaticamente incremente o index com uma unidade (e.g. *my-index-000005* depois de *my-index-04*)
  - Execução do Roll over:
    - Para fazer isso, antes de tudo criamos um alias para o index original, que já existe:

Listing 6.45 Creating an alias for an existing index

```
POST _aliases
{
  "actions": [
    {
      "add": {
        "index": "cars_2021-000001",
        "alias": "latest_cars_a",
        "is_write_index": true
      }
    }
  ]
}
```

Uses the _aliases API to invoke an add action
The set of actions
The add action
The index we wish to create for cars' rollover data
Creates the alias to the cars index
Makes the index writable

      - Deve ser writeable
      - Se o alias apontar para vários index, ao menos um deve ser writeable
    - Em seguida, utiliamos o endpoint _rollover **sobre o alias, não sobre o index**:

Listing 6.46 Rolling over the index

```
POST latest_cars_a/_rollover
```

    - Index novo então foi criado e o antigo virou read-only
  - Roll over faz três coisas:
    - Torna index antigo read-only
    - Cria o novo index com o nome incrementado, o resto permanece igual
    - Reaponta o alias para o novo index
  - Resta saber como automatizar isso:
    - Como fazer roll over quando o tamanho de shard ultrapassa um determinado limite?
    - Como fazer um index novo todo dia pra logs diários?

INDEX LIFECYCLE MANAGEMENT (ILM)

- Queremos gerir os index e agir sobre eles para otimizar uso de recursos (para mais ou para menos) conforme alterações no cluster vão acontecendo ao longo do tempo
- Podemos definir um conjunto de regras para fazer isso
  - E.g. regras para roll over.
    - Index atingiu um dado tamanho
    - Número de documentos atingiu um certo número
    - Virou o dia
- A vida de um index tem 5 fases:
  a. **Hot**

- Index operante e usado com frequência para read e write

b. **Warm**
   - Index é read-only e pode ser consultado

c. **Cold**
   - Index é read-only. Queries devem ser bem raras e devem ser devagar.

d. **Frozen**
   - Index é read-only. Queries são BEM raras e respostas BEM lentas.

e. **Delete**
   - Dados são apagados e memória é liberada
   - É comum tirar um snapshot do index antes de apagar, caso dados sejam necessários no futuro

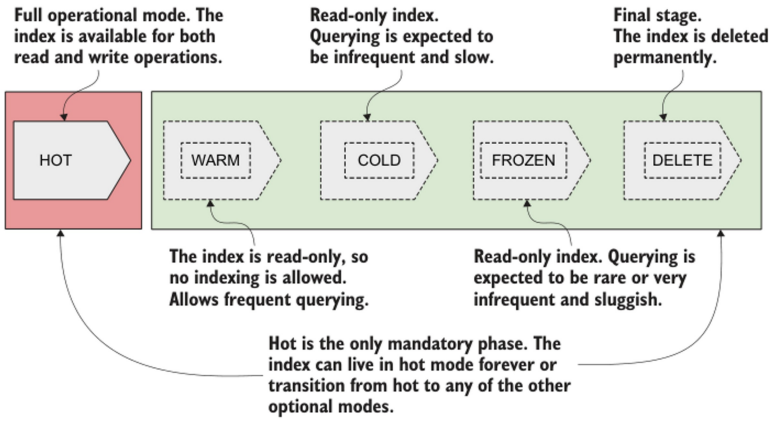

Figure 6.11 Lifecycle of an index

- Até então aprendemos como agir manualmente sobre os index, mas não como fazer isso com regras estabelecidas.
- Para isso, usamos API *_ilm*, em duas etapas:
  a. Definimos uma política:

Listing 6.48 Creating a policy with hot and delete phases

```
PUT _ilm/policy/hot_delete_policy
{
  "policy": {
    "phases": {
      "hot": {
        "min_age": "1d",
        "actions": {
          "set_priority": {
            "priority": 250
          }
        }
      },
      "delete": {
        "actions": {
          "delete" : { }
        }
      }
    }
  }
}
```


- No exemplo definimos duas fases só: hot e delete.
  - Index deve ser hot por 1d
    - Após isso, ele executa as actions (set priority, no caso) e passa para a fase delete
  - Então index se torna delete
    - Delete não tem min_age, então ocorre instantaneamente
    - Index é deletado

b. Associamo-la com um index:

Listing 6.49 Creating an index with an associated index lifecycle

```
PUT hot_delete_policy_index
{
  "settings": {                                  <- Uses the settings object to set the property
    "index.lifecycle.name":"hot_delete_policy" # Name of the policy
  }
}
```

Is It Just a Label?

No, **warm** and **frozen** phases are not just labels. They are actionable phases within an ILM policy, and Elasticsearch performs specific actions associated with each phase to manage resource usage effectively. However, the exact changes depend on the ILM policy configuration for the index. If no actions are defined for a phase in the ILM policy, then the phase change might act as a label only.

- Lifecycle com Rollover:
  - Podemos definir um rollover como action de uma política:

Listing 6.50 Simple policy definition for the hot phase

```
PUT _ilm/policy/hot_simple_policy
{
  "policy": {
    "phases": {
      "hot": {                                   <- Declares a hot phase
        "min_age": "0ms",                        <- The index enters this phase instantly.
        "actions": {
          "rollover": {                          <- The index rolls over if any of the conditions are met.
            "max_age": "1d",
            "max_docs": 10000,
            "max_size": "10gb"
          }
        }
      }
    }
  }
}
```

  - Neste exemplo, política é aplicada imediatamente, já que min_age é 0ms
  - Podemos também associar uma lifecycle policy com um template:
    - Necessário para a ILM ser aplicada ao rollover

Listing 6.51 Attaching a lifecycle policy to a template

```
PUT _index_template/mysql_logs_template
{
  "index_patterns": ["mysql-*"],                 <- Index pattern for all MySQL indexes
  "template":{
    "settings":{
      "index.lifecycle.name":"hot_simple_policy",   <- Attaches the policy

      "index.lifecycle.rollover_alias":"mysql-logs-alias"   <- Attaches an alias
    }
  }
}
```

    - Precisamos determinar a policy e o rollover_alias nesta ação
  - O último passo é criar um index que dê match no index_pattern definido no template:

Listing 6.52 Setting the index as writable for the alias

```
PUT mysql-index-000001                    <-- Creates an index with the appropriate format
{
  "aliases": {
    "mysql-logs-alias": {                 <-- Enables the alias
      "is_write_index":true               <-- The backing index must be writable.
    }
  }
}
```

- Logo:
  - Index terá a policy imediatamente
    - Será hot imediatamente
  - Permanece Hot até alguma condição estabelecida ocorrer
    - Quando houver alguma das condições, rollover ocorre
      - E sucessivamente para os novos index criados

- Condições das polcies são verificadas a cada 10minutos, por default
  - Pode ser alterado no settings do endpoint _cluster
  - Podemos resetar ciclo de varredura manualmente
  - Policies são uma background task, e esperam o próximo ciclo de der algum ruim, então pode não ocorrer exatamente no momento esperado.

Summary

- Elasticsearch exposes index APIs to create, read, delete, and update indexes.
- Every index has three sets of configurations: aliases, settings, and mappings.
- Indexes can be created implicitly or explicitly:
  - Implicit creation kicks in when an index doesn't exist and a document is indexed for the first time. Default configurations (such as one replica and one shard) are applied on an index that's created implicitly.
  - Explicit creation occurs when we instantiate indexes with a custom set of configurations using the index API.
- An index template lets us create indexes with predetermined configuration settings that, based on a matching name, are applied during index creation.
- An index can be resized using a shrinking or splitting mechanism. Shrinking reduces the number of shards, while splitting adds more primary shards.
- An index can be conditionally rolled over as required.
- Index lifecycle management (ILM) helps transition indexes between these lifecycle phases: hot, warm, cold, frozen, and delete. In the hot phase, the index is fully operational and open for searching and indexing; but in the warm and cold phases, the index is read-only.