Primeiros Passos Com Elasticsearch

Anael Carvalho
<anael.ferraz.carvalho@gmail.com>

Roteiro

1. Introdução
2. Conceitos básicos
3. Instalação
4. Interagindo
5. Indexando documentos
6. Failover e scaling
7. Buscas
8. Stack ELK
9. Kibana
10. Logstash
11. Elasticsearch em produção
12. Monitoramento
13. Referências

Repositório GitHub

http://bit.ly/1LpVa4f

Introdução

- Mecanismo de busca e análise
  - Textual, estruturada, analítica, etc
  - “Quase“ tempo real
- Código aberto
  - Baseado no Apache Lucene
  - Licença Apache 2
- Distribuído
- Altamente escalável

Introdução (cont.)

- “Esconde“ a complexidade do Lucene atrás de um pacote simplificado
  - API RESTful para comunicação
  - Configuração padrão sensata
  - Escalabilidade
- Ao mesmo tempo, flexível o suficiente para permitir configurações e funcionalidades avançadas
  - Controle de relevância, busca por proximidade, analisadores de línguas, etc
- Usado por Wikipedia, Stack Exchange, GitHub, etc

Conceitos básicos

- Orientado a *documentos*
  - Dados são guardados e indexados como *documentos*
  - JSON é usado como formato para serialização
  - A busca, pois, é feita por documentos baseada em seu conteúdo
- Qual a diferença para outras soluções NoSQL?
  - Todos os campos do documento são indexados por padrão (solução voltada para busca!)
    - Índice invertido

Conceitos básicos (cont.)

- Documentos são classificados com um *tipo*
  - Pode ser pensado como a “classe“ do documento
  - Cada *tipo* possui um nome (por ex. usuário) e um *mapeamento*
  - O *mapeamento* descreve a estrutura do documento (campos, tipo de dados, etc), algo como um schema!
- Documentos são guardados em *índices*
  - Análogo a um banco de dados!
  - Cada *índice* pode ter um ou mais *tipos*
  - Podemos pensar nos *tipos* como tabelas!

Conceitos básicos (cont.)

RDBMS	Elasticsearch
Banco de Dados	Índices
Tabelas	Tipos
Linhas	Documentos
Colunas	Campos do documento

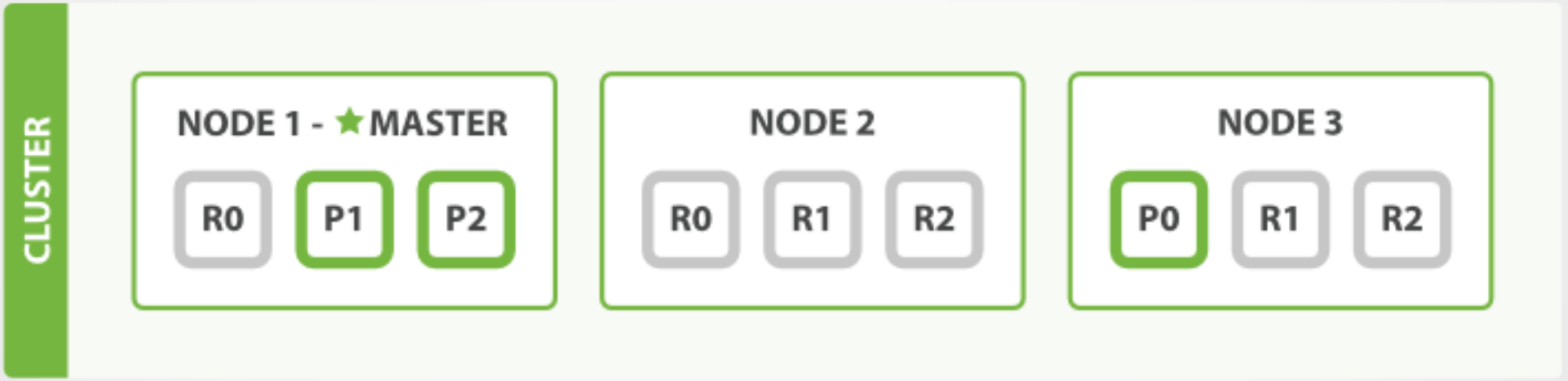
Conceitos básicos (cont.)

- Shards
  - Cada *índice* pode ter um ou mais *shards*
  - “Partição“ do índice que contém os dados (documentos)
    - Internamente, cada shard é uma instância do Lucene
  - Pode ser classificado como primário ou réplica
    - O número de *shards* primários é definido na criação do índice, não podendo mais ser alterado
    - O número de *shards* de réplica pode ser alterado a qualquer momento
    - Cada documento pertence a apenas um *shard* primário

Conceitos básicos (cont.)

- Nó
  - Uma instância do Elasticsearch
  - Armazena um ou mais *shards*
- Cluster
  - Grupamento de um ou mais *nós*
- Apenas um *nó* mestre existe por cluster
  - Responsável por tarefas “globais“
    - Criar/deletar índices, adicionar/remover nós do cluster, etc
  - Qualquer nó do cluster pode se transformar em mestre
- Podemos falar com qualquer nó do cluster, inclusive com o mestre

Conceitos básicos (cont.)



Instalação

DEMONSTRAÇÃO

Interagindo

- API RESTful
  - Padrão porta 9200
- API Java
  - Cliente “nó“
    - Cria uma instância do elasticsearch e se associa ao cluster, porém não armazena dados
  - Cliente “transporte“
    - Comunica-se remotamente com algum nó do cluster
  - Vantagens e desvantagens

Indexando documentos

- Utiliza-se a API de Indexação
  - Com id auto-gerado:
    - POST /{indice}/{tipo}
  - Com id externo
    - PUT /{indice}/{tipo}/{id}
- Documentos são imutáveis – para alterá-los é necessário substituir todo o conteúdo
  - Usa-se o endpoint com o id interno:
    - PUT /{indice}/{tipo}/{id}
  - Endpoint _update existe para facilitar o processo:
    - POST /{indice}/{tipo}/{id}/_update

Indexando documentos (cont.)

- Como saber se um documento existe?
  - Podemos utilizar o método HEAD
    - HEAD /{indice}/{tipo}/{id}
- Como garantir que um documento será criado em vez de atualizado?
  - Usa-se o endpoint _create:
    - PUT /{indice}/{tipo}/{id}/_create
- Como deletar um documento?
  - Método DELETE
    - DELETE /{indice}/{tipo}/{id}

Indexando documentos (cont.)

- Como garantir a consistência das atualizações?
  - Elasticsearch cria um campo de versão para cada documento (“_version“)
  - A aplicação pode passar como parâmetro a versão para aplicar as atualizações de dados
    - PUT /{indice}/{tipo}/{id}?version=1
  - Caso a versão tenha sido alterada, a requisição falha com status 409 Conflict

Indexando documentos (cont.)

DEMONSTRAÇÃO

Failover e scaling

- Basta iniciar novas instâncias do Elasticsearch para escalar
- Shards de réplica nunca ficam no mesmo nó que contém o shard primário
- Cluster automaticamente migra os shards entre os nós, mantendo os dados balanceados
- Em caso de falha de algum nó, os shards de réplica dos shards primários contidos no nó que falhou são promovidos a primários e novos shards de réplica são criados
  - Cluster é rebalanceado
- Em caso de falha do nó mestre, processo de eleição define o novo nó mestre

Failover e scaling

Status do Cluster	
Verde	Shards primários e de réplica ativos
Amarelo	Shards primários ativos porém nem todas as réplicas estão ativas. Em caso de falha de algum nó corremos risco de perda de dados!
Vermelho	Nem todos os shards primários estão ativos. Faltam dados aos índices!

Failover e scaling (cont.)

DEMONSTRAÇÃO

Buscas

- Funcionalidade principal da ferramenta
- Busca estruturada
  - Busca por algum campo específico ordenando resultados por algum outro campo
  - Similar a uma query SQL
- Busca textual
  - Encontrar documentos que contenham certas palavras-chaves, ordenando por relevância
- Busca mista
  - Combinação entre os dois anteriores

Buscas (cont.)

- Dois jeitos possíveis:
  - Busca via URI
    - Geralmente usada para testes ou buscas simples
    - Opções limitadas
    - GET /{indice}/{tipo}/_search?q={query}
  - Busca via Request Body
    - Utiliza Query DSL
      - Linguagem de query específica do Elasticsearch
      - Define uma série de queries e filtros e permite a combinação dos mesmos
    - POST|GET /{indice}/{tipo}/_search

Buscas (cont.)

- Filtro
  - Usado para buscar campos que contém valores exatos
  - Buscas que podem ser respondidas com sim ou não
  - Exemplos: terms, range, exists, bool, etc
- Query
  - Usado para buscas textuais
  - Buscas cujo resultado dependam de relevância
  - Exemplos: match, bool, etc

Buscas (cont.)

- Analisadores
  - Quando um documento é indexado, os campos texto são “quebrados“ em termos que podem ser buscados
  - A “quebra“ é feita por um analisador
    - É possível criar novos ou usar algum pré-definido
    - Analisador padrão é aplicado a todos os campos textos caso nenhum seja definido
      - A associação é feita no *mapeamento* dos *tipos*
  - Posteriormente, a busca feita no campo deve passar pelo mesmo analisador para garantir a consistência dos resultados
- Paginação
  - Padrão de 10 resultados por página

Buscas (cont.)

- Ordenação
  - Feita por algum campo específico ou por relevância
- Relevância
  - Cada documento voltará com um campo especial chamado “_score“ que contém um decimal representando a relevância
  - Quanto maior o número, maior a relevância
  - Relevância é calculada de acordo com um algoritmo de similaridade
  - O algoritmo padrão utilizado é o Term Frequency/Inverse Document Frequency (TF/IDF)

Buscas (cont.)

DEMONSTRAÇÃO

Stack ELK

- O Elasticsearch pode ser utilizado em conjunto com outras duas ferramentas, que juntas formam o stack ELK
  - Elasticsearch
  - Logstash
  - Kibana

Kibana

- Plataforma de análise e visualização de dados
- Podemos criar e armazenar queries customizadas, visualizações de acordo com widgets pré-definidos, e dashboards para visualização de dados em tempo real
- Fácil configuração e utilização

Kibana

DEMONSTRAÇÃO

Logstash

- Ferramenta para coleta, processamento e entrega de dados de fontes variadas para locais diversos
- Entradas
  - Arquivos, HTTP, TCP, stdin, SQS, Twitter, Redis, etc
- Filtros
  - CSV, XML, JSON, Grok, etc
    - Mais de 120 expressões Grok pré-definidas para parsing de entradas
- Saídas
  - Elasticsearch, stdout, S3, arquivos, MongoDB, etc

Logstash

DEMONSTRAÇÃO

Elasticsearch em produção

- Preferir memória a capacidade de processamento
  - 8GB a 64GB
- Disco quase sempre será o gargalo, usar SSD se possível
- Nomear cluster e nós
- Mudar diretório de armazenamento de dados
- Desabilitar multicast
- No mínimo 3 nós
  - Tentar evitar a situação de “split brain“
  - Setar o parâmetro “minimum_master_nodes“ de modo a ter um quórum de metade mais um do número total de nós

Elasticsearch em produção

- JVM – Dar apenas metade da memória disponível
  - A outra metade será usada pelo Lucene, que consome do SO
  - Não ultrapassar 32GB
- Desabilitar ou reduzir swap
- Aumentar o número de file descriptors (64.000)
- Para grande quantidade de nós, verificar configurações de recuperação

Monitoramento & Troubleshooting

- Plugin Marvel
- Saúde do cluster
  - GET _cluster/health
- Saúde dos índices
  - GET _cluster/health?level=indices
- Saúde dos shards
  - GET _cluster/health?level=shards

Monitoramento & Troubleshooting

- Status dos nós
  - GET _nodes/stats
- Status do cluster (agregação dos nós)
  - GET _cluster/stats
- Status dos índices
  - GET {indice}/_stats

Referências

- https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html
- https://www.elastic.co/guide/en/kibana/current/index.html
- https://www.elastic.co/guide/en/logstash/current/index.html
- Elasticsearch, The Definitive Guide (Clinton Gormley & Zachary Tong – O'Reilly Media – ISBN 9781449358549)

PERGUNTAS

FIM