нетология

Elasticsearch

Роман
Гордиенко

Роман Гордиенко

Backend Developer, sdvor.com

Роман Гордиенко

План занятия

1. Историческая справка
2. Текущая версия
3. PACELC-теорема
4. Архитектура хранения данных
5. Конфигурирование
6. Api мониторинга состояния
7. Архитектура кластера
8. Бэкапы и восстановление данных
9. X-PACK
10. Итоги
11. Домашнее задание

Историческая справка

Историческая справка

В **2004 году Шай Бейнон** (Shay Baynon) разработал систему **Compass**, которая представляла простое API для работы с Java Search Engine под названием **Lucene**.

При разработке третьей версии Compass, было установлено, что для создания масштабируемого решения, нужно переписать систему “с нуля”.

Так в **феврале 2010 года** была выпущена первая версия **Elasticsearch** - масштабируемая поисковая система, поддерживающая многопоточность.

Историческая справка

Elasticsearch является свободно-распространяемым движком для поиска и аналитики, предоставляющим RESTful интерфейс взаимодействия.

Elasticsearch относится к NoSQL базам данных.

Самые популярные примеры использования Elasticsearch:

- хранение логов и их анализ;
- сбор и агрегирование различных пользовательских данных;
- полнотекстовый поиск;
- сбор метрик, временных рядов и событий.

Текущая версия

Текущая версия

Текущая версия **Elasticsearch 7.9.0** была выпущена **2020-08-18**.

Подробно ознакомиться можно тут: https://www.elastic.co/guide/en/elasticsearch/reference/current/release-highlights.html

Взято с сайта: elasticsearch.co

PACELC-теорема

PACELC-теорема

В системе может быть нарушение согласованности,
в пользу доступности данных при сетевом разделении.
В случае штатной работы - в пользу задержки ответа.

Это PA+EL система.

Архитектура хранения данных

Архитектура хранения данных

Данные в Elasticsearch организованы в индексы.

Каждый индекс состоит из одного или нескольких сегментов (шардов - shard).

По мере того, как данные записываются в шард - они периодически публикуются в иммутабельном виде на диске, и именно в это время они становятся доступными для запросов.

Поскольку сегменты иммутабельны - обновление документа требует, чтобы Elasticsearch сначала нашел существующий документ, затем пометил его как удаленный и добавил обновленную версию.

В Elasticsearch каждый запрос выполняется в одном потоке для каждого шарда. Однако несколько шардов могут обрабатываться параллельно.

Архитектура хранения данных

Существует два типа шардов:

- primary;
- replicas/secondary.

Каждый документ в индексе принадлежит одному первичному шарду.

Шард-реплика - это копия primary шарда.

Реплики являются копиями ваших данных для защиты от сбоев оборудования и увеличения скорости работы с данными.

Архитектура хранения данных

Статусы ЖЦ шардов принимают значения:

- INITIALIZING - шард в процессе восстановления и индексации;
- RELOCATING - шард в процессе релокации на другой узел;
- STARTED - шард запущен;
- UNASSIGNED - шард не привязан ни к одной из нод.

Конфигурирование

Конфигурирование

Для тонкой настройки Elasticsearch имеются 3 файла конфигурации:

- **elasticsearch.yml** - для настройки работы Elasticsearch;
- **jvm.options** - для настройки Elasticsearch JVM;
- **log4j2.properties** - для настройки логгирования Elasticsearch.

Конфигурирование

Настройка через environment-переменные:

Все настройки можно производить через **env** -переменные, если указать в elasticsearch.yml значение ключа в нотации ${}.

Например:

node.name: ${HOSTNAME}

Конфигурирование

Основные разделы настройки elasticsearch.yml:

- Cluster
- Node
- Index
- Paths
- Plugin
- Memory
- Network And HTTP
- Gateway
- Recovery Throttling
- Discovery

Конфигурирование

Cluster

Основная настройка раздела “Cluster” - **cluster.name**

Имя кластера идентифицирует ваш кластер для auto-discovery (широковещательные запросы в рамках сети).

Если вы одновременно запустили несколько кластеров в одной сети, следует сделать им различные имена.

Конфигурирование

Node

Основные настройки раздела “Node”:

- **node.name** - string, имя текущей ноды;
- **node.master** - bool, запуск ноды в режиме “master”;
- **node.data** - bool, запуск ноды в режиме “data”.

По умолчанию узел запускается в режиме master-data.

Конфигурирование

Master	Data	Спецификация узла
True	True	Настройка по умолчанию. Узел является координатором кластера и хранит данные.
True	False	Узел является “координатором” кластера и не хранит данные
False	False	Узел является “вспомогательным” - выполняет операции над данными

Конфигурирование

Index

Основные настройки раздела “Index”:

- **index.number_of_shards** - количество “шардов” данных, по умолчанию 5;
- **index.number_of_replicas** - количество реплик данных, по умолчанию 1.

Best practice:

- **index.number_of_shards = number_of_nodes * 3**;
- **index.number_of_replicas = number_of nodes - 1**.

Такая конфигурация позволит вам осуществить быстрый старт кластера с оптимизированными индексами.

Конфигурирование

Paths

Основные настройки раздела “Paths”:

- **path.conf** - путь до конфигурационных файлов;
- **path.data** - директория для хранения индексов;
- **path.work** - директория для хранения временных файлов;
- **path.logs** - директория хранения логов работы.

Конфигурирование

Network and HTTP

Основные настройки раздела “Network and HTTP”:

- **network.bind_host** - адрес привязки узла;
- **network.publish_host** - адрес, по которому с данным узлом можно связаться;
- **network.host** - одновременная конфигурация bind_host и publish_host;
- **transport.tcp.port** (установка кастомного порта для обмена, 9300 - дефолт;
- **http.port** - установка кастомного порта http траффика, 9200 - дефолт;
- **http.max_content_length** - максимальный размер тела http запроса.

Конфигурирование

Discovery

Discover контролирует нахождение узлов в кластере и наличие ведущего(их) узла(ов). По умолчанию используется многоадресное обнаружение (Multicast).

Основные настройки раздела “Discovery”:

- **discovery.zen.minimum_master_nodes** - минимальное количество мастер нод, при котором кластер считается работоспособным;
- **discovery.zen.ping.timeout** - время ожидания ответа от других нод в кластере;
- **discovery.zen.ping.multicast.enabled** - контроль включения многоадресного обнаружение;
- **discovery.zen.ping.unicast.hosts** - перечисление адресов узлов кластера в точечном обнаружении (Unicast).

Пример конфигурации: **["host1", "host2:port"]**

Конфигурирование

Основные настройки jvm.options:

- **Heapsize**;
- **GC**;
- **GC logging**.

Конфигурирование

Heapsize

Xms

Xmx

- выставляются одинаковые размеры;
- минимум - 1гб;
- максимум - не более 50% RAM сервера.

Конфигурирование

Garbage Collector

Выбирается один из:

- ***- XX:+UseSerialGC****;*
- ***-XX:+UseParallelGC****;*
- ***-XX:+UseConcMarkSweepGC****;*
- ***-XX:+UseG1GC****.*

Выбирается экспериментально под задачу на основе профилирования использования Java Heap.

API мониторинга состояния

API мониторинга состояния

Elasticsearch предоставляет удобное API для мониторинга состояния кластера.

Основные методы для наблюдения за состоянием Elasticsearch:

- **_cluster/health** - состояние кластера;
- **_cat/indices** - состояние индексов;
- **_cat/shards** - состояние шардов.

API мониторинга состояния

Основные параметры результата вызова _cluster/health:

- status;
- number_of_nodes;
- number_of_data_nodes;
- relocating_shards;
- initializing_shards;
- unassigned_shards;
- number_of_pending_tasks;
- task_max_waiting_in_queue_millis;
- active_shards_percent_as_number.

API мониторинга состояния

Пример результата вызова метода _cluster/health

```
{
  "cluster_name" : "testcluster",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 1,
  "active_shards" : 1,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 1,
  "delayed_unassigned_shards": 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch": 0,
  "task_max_waiting_in_queue_millis": 0,
  "active_shards_percent_as_number": 50.0
}
```

Взято с сайта: https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-health.html

API мониторинга состояния

Status

Состояние работоспособности кластера на основе состояния его шардов.

Принимает значения:

- **red** - один или несколько primary шард unassigned;
- **yellow** - все primary шарды в состоянии assigned. Часть secondary - шард в состоянии unassigned;
- **green** - все шарды в состоянии assigned.

API мониторинга состояния

Number_of_nodes/Number_of_data_nodes

Количество узлов кластера и количество узлов, выделенных для записи.

Важно мониторить изменение данной величины, для обнаружения сбоев в работе узлов.

API мониторинга состояния

Relocating_shards

Количество шард в состоянии RELOCATED.

При активном перемещении шард, нужно чтобы тренд данной величины был монотонно убывающим.

API мониторинга состояния

Initializaing_shards

Количество шард в состоянии INITIALIZED.

Данная величина при нагрузке должна быть константой.

В случае ее увеличения - кластер не успевает распределять шарды по узлам.

API мониторинга состояния

Unassigned_shards

Количество шард в состоянии Unassigned.

При появлении значения данной величины отличной от 0 может говорить о нарушениях в работе кластера.

API мониторинга состояния

Number_of_pending_tasks

Количество задач в состоянии ожидания. Например чтение из индекса.

При появлении значения данной величины отличной от 0 говорит о том, что кластер не справляется с нагрузкой.

API мониторинга состояния

Task_max_waiting_in_queue_millis

Время ожидания задачи в очереди (в миллисекундах).

В идеале должно быть нулевым.

Иначе - кластер не справляется с нагрузкой.

API мониторинга состояния

Active_shards_percent_as_number

Количество всех активных шардов в процентах.

Активные шарды - это те, что находятся не в состоянии unassigned/initializing.

Отклонения данной величины от 100% говорит о деградации кластера.

API мониторинга состояния

Основные параметры результата вызова _cat/indices:

- **health** - обобщенное состояние индекса red/yellow/green;
- **status** - текущий статус индекса;
- **index** - название индекса;
- **uuid** - уникальный идентификатор индекса;
- **pri** - количество primary шард индекса;
- **rep** - количество реплик шард индекса;
- **docs.count** - количество документов в индекса;
- **docs.deleted** - количество документов в индексе в статусе deleted;
- **store.size** - размер store;
- **pri.store.size** - размер primary store.

API мониторинга состояния

Пример результата вызова метода _cat/indices

```
health status index           uuid                   pri rep docs.count docs.deleted store.size pri.store.size
yellow open   my-index-000001 u8FNjxh8Rfy_awN11oDKYQ   1   1       1200            0     88.1kb         88.1kb
green  open   my-index-000002 nYFWZEO7TUiOjLQXBaYJpA   1   0          0            0       260b           260b
```

Взято с сайта: https://www.elastic.co/guide/en/elasticsearch/reference/current/cat-indices.html

API мониторинга состояния

Основные параметры результата вызова _cat/shards:

- имя индекса;
- номер реплики;
- обозначение привязки шарда к реплике p(primary)/r(replica);
- состояние шарда;
- опционально (при нарушениях работы): развернутое состояние шарда;
- количество документов в шарде;
- размер шарда (в байтах);
- хост, на котором размещен шард;
- id шарда.

API мониторинга состояния

Пример результата вызова метода _cat/shards

```
my-index-000001 0 p STARTED    3014 31.1mb 192.168.56.10 H5dfFeA
my-index-000001 0 r STARTED    3014 31.1mb 192.168.56.30 bGG90GE
my-index-000001 0 r STARTED    3014 31.1mb 192.168.56.20 I8hydUG
my-index-000001 0 r UNASSIGNED ALLOCATION_FAILED
```

Взято с сайта: https://www.elastic.co/guide/en/elasticsearch/reference/current/cat-shards.html

API мониторинга состояния

Представленное API можно вызвать также, указывая например:

- параметры сортировки;
- вид выходных данных;
- конкретизируя индекс или шард (доступны wildcard).

Подробнее ознакомиться с доступными методами можно в официальной документации на REST API Elasticsearch.

Архитектура кластера

Архитектура кластера

Кластер Elasticsearch обеспечивает работоспособность, даже если некоторые из его компонентов вышли из строя.

Кластера разделяются на:

- One-node cluster;
- Two-node cluster;
- Two-node cluster with tiebraker;
- Three-node or more cluster.

Архитектура кластера

One-node cluster

Не является устойчивым.

Нужно делать постоянные бекапы данных.

Количество реплик индексов = 0.

Не рекомендуется в промышленном использовании.

Архитектура кластера

Two-node cluster

Обе ноды рекомендуется запускать в режиме data_node.

Количество реплик индексов = 1.

Одна из нод должна быть master=true, другая master=false. Это позволяет избежать проблем кворума.

Нужно посылать запросы сразу на оба узла, чтобы сохранить согласованность данных.

Не рекомендуется для использования в промышленной среде.

Архитектура кластера

Two-node cluster with tiebraker

Аналогично Two-node cluster, но добавляется третий узел в режиме master=true, data=false.

Таким образом, мы избежим проблем кворума в кластере при падении одной из master-нод.

Также нам не нужно производить вертикальное масштабирование, т.к. третий узел в режиме data=false.

Так как не решается проблема с реплицированием и согласованностью данных - не рекомендуется к промышленному использованию.

Архитектура кластера

Three-node or more cluster

Все ноды запускаются в режимах master=true, data=true.

Количество реплик индексов >= 2.

Нужно посылать запросы сразу на все узлы, чтобы сохранить согласованность данных.

Такой кластер устойчив к выходу из строя как минимум 1 узла.

Рекомендуется использовать в промышленной среде.

Архитектура кластера

Критерии выбора архитектуры для промышленного использования:

- Состояние работоспособности кластера - green
- Есть как минимум два узла в режиме data=true
- Есть как минимум 1 реплика данных у каждого индекса
- Кластер имеет не менее трех узлов, не менее двух из них master=true
- Клиенты настроены на отправку своих запросов более чем на один узел.

Бэкапы и восстановление данных

Бэкапы и восстановление данных

Основные термины в Elasticsearch, относящиеся к восстановлению данных:

- **Repository** - место, где хранятся бэкапы. В одном репозитории можно хранить несколько бэкапов. Repository содержит snapshot только 1 кластера.
- **Snapshot** - это и есть сам бэкап.

Бэкапы и восстановление данных

Жизненный цикл Snapshot выглядит следующим образом:

- создаем repository для snapshot (либо можно использовать существующий);
- создаем snapshot данных;
- наблюдаем за состоянием snapshot;
- восстанавливаем данные из snapshot;
- наблюдаем прогресс восстановления данных;
- удаляем snapshot.

Вызовы API работы со Snapshot довольно просты и хорошо описаны в официальной документации Elasticsearch.

X-PACK

X-PACK

X-Pack - это расширение для Elastcisearch, которое предоставляет:

- безопасность доступа к данным;
- оповещения;
- мониторинг;
- отчетность;
- машинное обучение.

Доступен в установке Elasticsearch по умолчанию.

Итоги

Итоги

Elasticsearch - мощный и отказоустойчивый инструмент, позволяющий решать большой круг задач.

В текущей лекции мы узнали, что **Elasticsearch:**

- PA+EL система;
- хранит данные в индексах и шардах;
- имеет множество настроек для оптимизации своей работы;
- предоставляет REST API для мониторинга своего состояния;
- имеет best practice по кластеризации;
- обладает встроенным механизмом бэкапа и восстановления.

Домашнее задание

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задавайте **в чате** мессенджера Slack.
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

нетология

Задавайте вопросы и пишите отзыв о лекции!

Роман Гордиенко

Роман Гордиенко