

индексирование, анализ, поиск и агрегирование данных

Elasticsearch

для разработчиков

Анураг Шриваастава



2-е

издание



Elasticsearch 8 for Developers

2nd Edition

*A beginner's guide to indexing, analyzing,
searching, and aggregating data*

Anurag Srivastava



www.bpbonline.com

Elasticsearch для разработчиков

2-е издание

*индексирование, анализ, поиск
и агрегирование данных*

Анураг Шривастава



Санкт-Петербург · Москва · Минск

2025

Выпущено
при поддержке **КРОК**

ББК 32.973.233.02

УДК 004.65

Ш86

Шриавастава Анураг

- Ш86 Elasticsearch для разработчиков: индексирование, анализ, поиск и агрегирование данных. 2-е изд. — СПб.: Питер, 2025. — 336 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-4211-8

Elasticsearch — мощный инструмент для работы с большими объемами данных. Это масштабируемая, надежная и быстрая система со множеством функций для анализа и поиска данных.

Книга представляет собой подробное руководство по использованию Elasticsearch для управления данными. Вначале приводится обзор Elasticsearch, где описана его важность в современном мире. Далее рассматриваются основы Elasticsearch, включая установку, настройку и управление индексами. Затем автор переходит к более сложным темам, таким как обработка геопространственных данных и использование агрегаций для анализа данных. Кроме того, внимание удалено вопросам оптимизации работы, производительности и администрирования. Практические примеры помогут понять и применить изученные концепции.

Вы получите глубокое представление об Elasticsearch и сможете использовать его для управления большими объемами данных и извлечения из них ценной информации.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.233.02

УДК 004.65

Права на издание получены по соглашению с BPB Publications, India. Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Информация, содержащаяся в данной книге, получена из источников, рассматриваемых издательством как надежные. Тем не менее, имея в виду возможные человеческие или технические ошибки, издательство не может гарантировать абсолютную точность и полноту приводимых сведений и не несет ответственности за возможные ошибки, связанные с использованием книги. В книге возможны упоминания организаций, деятельность которых запрещена на территории Российской Федерации, таких как Meta Platforms Inc., Facebook, Instagram и др. Издательство не несет ответственности за доступность материалов, ссылки на которые вы можете найти в этой книге. На момент подготовки книги к изданию все ссылки на интернет-ресурсы были действующими.

ISBN 978-9355519825 англ.

© 2024 BPB Publications, India. All rights reserved.

First published in the English language under the title Elasticsearch 8 for Developers by Anurag Srivastava. ISBN: 9789355519825 by BPB Publications India (sales@bpbonline.com)

© Перевод на русский язык ООО «Прогресс книга», 2025

© Издание на русском языке, оформление ООО «Прогресс книга», 2025

© Серия «Библиотека программиста», 2025

ISBN 978-5-4461-4211-8

Оглавление

Об авторе.....	16
О научном редакторе	17
Благодарности	18
Предисловие	19
От издательства.....	22
О научном редакторе русского издания	22
Глава 1. Начало работы с Elasticsearch.....	23
Введение.....	23
Структура главы.....	24
Цели.....	24
Введение в поиск данных.....	24
Что такое Elasticsearch и почему он важен для поиска и аналитики.....	26
Обзор архитектуры и компонентов Elasticsearch.....	27
Узел	27
Кластер	33
Шарды.....	35
Документы.....	36
Области применения и варианты использования Elasticsearch.....	37
Поиск данных	37
Регистрация и анализ данных	38
Мониторинг производительности приложений	38
Мониторинг производительности системы.....	39
Визуализация данных	39

6 Оглавление

Клиенты Elasticsearch и сценарии их использования.....	41
Java	41
PHP	42
Perl.....	42
Python.....	42
.NET	43
Ruby	43
JavaScript.....	43
Заключение	44
Вопросы.....	44
Глава 2. Установка Elasticsearch	45
Введение.....	45
Структура	45
Цели.....	46
Введение в Elasticsearch 8	46
Повышение производительности индексирования	46
Повышение производительности поиска.....	46
Улучшения межклUSTERного поиска.....	46
Усовершенствования в области безопасности	47
Усовершенствования в области эксплуатации	47
Установка Elasticsearch в Linux и macOS	47
Установка Elasticsearch в Linux.....	47
Установка Elasticsearch в macOS	48
Установка Elasticsearch с помощью пакета Debian	49
Установка пакета Debian вручную	49
Установка Elasticsearch с помощью пакета RPM.....	50
Установка пакета RPM вручную	51
Установка Elasticsearch в OC Windows.....	51
Запуск и проверка службы Elasticsearch	52
Elasticsearch REST API.....	52
CAT API.....	53
Параметры CAT API	53
Verbose.....	53

Help	54
Заголовки.....	55
Форматы ответов	55
Сортировка.....	57
CAT COUNT API	57
CAT HEALTH API	57
CAT INDICES API	58
CAT MASTER API	58
CAT NODES API.....	59
CAT SHARDS API	59
API кластеров.....	60
API состояния кластера.....	60
API статистики кластера	60
Заключение.....	63
Вопросы.....	63
Глава 3. Elastic Stack: экосистема Elasticsearch.....	64
Введение.....	64
Структура	64
Цели.....	65
Обзор компонентов Elastic Stack	65
Elasticsearch: механизм поиска и аналитики.....	66
Logstash: пайплайн обработки данных	67
Плагины ввода данных Logstash.....	69
Плагины фильтрации Logstash.....	69
Плагины вывода данных Logstash.....	70
Kibana: инструмент визуализации данных.....	74
Beats: легковесная отправка данных	78
Filebeat	79
Metricbeat.....	83
Packetbeat.....	85
Winlogbeat.....	86
Auditbeat.....	87
Heartbeat	88
Functionbeat	90

8 Оглавление

Интеграция компонентов Elastic Stack.....	91
Получение логов Apache с помощью Logstash.....	92
Заключение	94
Вопросы.....	94
Глава 4. Подготовка данных к индексированию	95
Введение.....	95
Структура	95
Цели.....	96
Важность подготовки данных перед индексированием	96
Анализаторы Elasticsearch.....	96
Встроенные анализаторы	102
Стандартный анализатор	103
Простой анализатор	103
Анализатор пробелов.....	104
Анализатор стоп-слов.....	104
Анализатор ключевых слов	105
Анализатор шаблонов.....	105
Языковые анализаторы.....	106
Анализатор отпечатков	107
Пользовательские анализаторы.....	107
Токенизаторы Elasticsearch	107
Словесно-ориентированные токенизаторы	108
Фильтры токенов	116
Фильтры символов в Elasticsearch.....	117
Фильтр HTML strip	117
Фильтр сопоставления.....	117
Фильтр символов для замены шаблона	117
Нормализаторы	118
Заключение	119
Вопросы.....	119
Глава 5. Импорт данных в Elasticsearch	120
Введение.....	120
Структура	120
Цели.....	120

Почему данные важны для бизнеса.....	121
Доставка данных	122
Поглощение данных.....	122
Хранение данных	124
Визуализация данных	125
Импорт данных в Elasticsearch с помощью Beats	127
Filebeat	128
Metricbeat.....	134
Packetbeat.....	139
Заключение.....	149
Вопросы.....	149
Глава 6. Управление индексами: создание, обновление и удаление индексов Elasticsearch	150
Введение.....	150
Структура	150
Цели.....	151
Введение в создание и маппинг индексов Elasticsearch	151
Создание индекса без документов	151
Создание индекса, содержащего документы.....	153
Получение маппинга индекса	154
Создание маппинга.....	155
Управление индексами в Elasticsearch	156
Выполнение операций над индексами Elasticsearch	158
Закрытие индекса	159
Удаление индекса	159
Замораживание индекса	159
Обновление индекса	159
Принудительное слияние индекса.....	160
Очистка кэша индекса.....	160
Сброс индекса на диск.....	160
Добавление политики жизненного цикла	160
API Elasticsearch для работы с индексами	161
Управление индексами.....	162
Настройки индекса.....	174

10 Оглавление

Управление шаблонами индексов Elasticsearch.....	177
Создание шаблона индекса.....	177
Управление жизненным циклом индекса в Elasticsearch	180
Заключение.....	181
Вопросы.....	181
Глава 7. Возможности поиска: освоение Query DSL и техник поиска	182
Введение.....	182
Структура	183
Цели.....	183
Поиск по URI	183
Пустой поиск	184
Поиск по полю.....	184
Query DSL	188
Фильтры и запросы.....	189
Запрос.....	190
Типы запросов	191
Полнотекстовый поиск	192
Запросы на уровне терминов	197
Сложные запросы	201
Мультипоиск	204
API мультипоиска.....	204
Шаблоны поиска и мультипоиска	205
Шаблон поиска	205
Шаблон мультипоиска	207
Explain API.....	207
Обратная частота документа и частота термина.....	210
Profile API	211
Заключение.....	213
Вопросы.....	214
Глава 8. Работа с геоданными в Elasticsearch	215
Введение.....	215
Структура	216
Цели.....	216

Введение в геопространственный поиск	216
Типы геоданных в Elasticsearch	217
Данные типа геоточка.....	218
Создание маппинга.....	219
Сохранение геоточек.....	219
Данные типа геоформа.....	220
Создание маппинга.....	221
Сохранение данных геоточки	222
Геозапросы и фильтры DSL.....	228
Запросы георасстояния.....	229
Геополигональные запросы	230
Границные запросы	232
Запросы геоформ.....	233
Примеры использования.....	234
Поиск ресторана	235
Геоагрегация.....	239
Заключение.....	241
Вопросы.....	242
Глава 9. Анализ данных с помощью агрегаций Elasticsearch	243
Введение.....	243
Структура	244
Цели.....	244
Введение в агрегацию Elasticsearch.....	244
Агрегация бакетов	245
Агрегация диапазонов	248
Составная агрегация	251
Агрегация терминов	255
Агрегация фильтра.....	257
Агрегация фильтров	258
Агрегация георасстояния	259
Агрегация метрик	261
Поиск минимума	261
Поиск максимума.....	262
Вычисление среднего.....	262

Вычисление суммы.....	263
Подсчет количества значений	264
Сбор статистики	264
Сбор расширенной статистики	265
Вычисление процентиелей	267
Агрегация матриц	268
Сбор матричной статистики.....	268
Агрегация пайплайнов	270
Вычисление среднего значения по бакетам.....	271
Вычисление максимального значения по бакетам.....	274
Вычисление суммы значений по бакетам.....	276
Заключение.....	277
Вопросы.....	278
Глава 10. Настройка производительности.....	279
Введение.....	279
Структура	279
Цели.....	280
Стратегии оптимизации производительности Elasticsearch	280
Оптимизация Elasticsearch для работы с большими объемами данных	281
Настройка скорости индексирования Elasticsearch	282
Массовые запросы вместо одного	282
Разумное использование кластера Elasticsearch.....	285
Увеличение интервала обновления	285
Отключение репликации.....	286
Использование автоматически генерируемых идентификаторов	286
Настройка размера буфера индексирования.....	287
Использование более быстрого оборудования.....	287
Выделение памяти для кэша файловой системы.....	287
Настройка скорости поиска в Elasticsearch	288
Моделирование документов	288
Поиск по меньшему количеству полей	288
Предварительное индексирование данных.....	290

Маппинг идентификаторов в качестве ключевых слов.....	292
Принудительное слияние индексов, доступных только для чтения.....	293
Использование фильтра вместо запроса.....	293
Увеличение количества реплик.....	293
Извлечение только необходимых полей.....	294
Использование более быстрого оборудования.....	295
Выделение памяти для кэша файловой системы.....	295
Отказ от стоп-слов при поиске.....	295
Отказ от скриптовых запросов.....	296
Настройка Elasticsearch для использования диска.....	297
Сжатие индекса.....	297
Принудительное слияние.....	298
Отключение необязательных функций.....	299
Оптимизация типов числовых полей	303
Лучшие практики Elasticsearch.....	303
Явное определение маппинга индексов Elasticsearch	303
Оптимизация производительности кластера Elasticsearch.....	304
Как избежать проблемы split-brain	305
Включение лога медленных запросов.....	306
Заключение.....	307
Вопросы.....	308
Глава 11. Администрирование: управление кластерами Elasticsearch.....	309
Введение.....	309
Структура	310
Цели.....	310
Безопасность Elasticsearch.....	310
Настройка TLS.....	311
Пароли кластера Elasticsearch	311
Настройка доступа на основе ролей с помощью Kibana	315
Псевдонимы индекса.....	319
Создание репозитория и снапшота	321
Создание репозитория	322
Создание снапшота.....	322

14 Оглавление

Восстановление из снапшота	324
Общая схема Elastic	324
Зачем нужна общая схема?.....	325
Введение в схему ECS	326
Общие рекомендации ECS	326
Рекомендации по именованию полей ECS	327
Начало работы с ECS.....	328
Масштабирование кластера Elasticsearch	329
Вертикальное масштабирование	329
Горизонтальное масштабирование.....	330
Мониторинг Elasticsearch	331
Заключение	333
Вопросы.....	334

*Посвящается
моим любимым родителям:
Вирендре Натху Шриваставе
Киран Шриваставе
и
моей жене Чанчал
и
детям Анвиту и Адитри*

Об авторе

Анураг Шривастава работает в сфере ИТ уже более 16 лет и имеет успешный опыт ведения различных государственных и частных проектов. Он занимал ключевые должности в области разработки и технического управления, позволявшие ему демонстрировать свою адаптивность и умение достигать успеха.

В сфере профессиональных интересов Анурага лежит и литература: он автор нескольких высоко оцененных книг. Его публикации широко известны благодаря представленному в них комплексному и практическому подходу к Kibana и Elasticsearch.

Помимо написания книг, Анураг проводит корпоративные тренинги по эффективному использованию Elasticsearch, Kibana, ELK и Cumulocity IoT.

Обладая разнообразным набором навыков, сильными лидерскими качествами и страстью к инновациям, Анураг Шривастава делает большой вклад в ИТ-инициативы. Он стремится добиваться исключительных результатов и быть в курсе тенденций и достижений в отрасли. Анураг — проверенный эксперт, помогающий совершенствовать сферу технологий.

О научном редакторе

Джива — сооснователь компании Epsil Technologies Private Limited, которая специализируется на предоставлении высококачественной поддержки и решений для Elasticsearch и ее экосистемы. Под эгидой Epsil он **собрал сплоченную группу** высококвалифицированных сертифицированных инженеров Elasticsearch.

Он помогает другим компаниям использовать возможности Elastic в различных сценариях, включая корпоративные поисковые системы, решения для мониторинга, оптимизацию кластеров и обновление версий.

Джива обладает обширным опытом проектирования, создания и устранения неисправностей кластеров Elasticsearch в различных средах, включая локальные, облачные системы и системы **Elastic Cloud Enterprise (ECE)**.

Благодаря своему богатому опыту он не только хорошо разбирается в Elastic (и стеке ELK), но и с большим удовольствием делится информацией о последних версиях и возможностях стека Elastic.

Он также получил бронзовую награду Elastic Contributor — 2022 в знак признания своего вклада в развитие продукта и продуктового сообщества.

В центре вселенной Дживы — любимая жена Сандхия и двое их необыкновенных детей, Дхияра и Аадхира, и он дорожит каждым моментом, который они проводят вместе.

Благодарности

Я глубоко благодарен своей семье и друзьям за их неизменную поддержку и за то, что вдохновили меня на написание этой книги. Особенно я благодарен своим родителям, своей жене Чанчал и своим детям, Анвиту и Адитри, за их терпение и понимание во время этой работы.

Я также благодарен издательству BPB Publications за руководство и опыт в подготовке книги к изданию. Это был долгий путь, пройденный при участии рецензентов, технических экспертов и редакторов, оказавших мне неоценимую поддержку.

Я также хотел бы отметить ценный вклад своих коллег и сослуживцев, которые на протяжении многих лет работы в технологической отрасли многому меня научили и давали полезную обратную связь.

Наконец, я благодарен всем читателям, которые проявили интерес к моей книге и способствовали ее появлению на свет. Ваша поддержка бесцenna.

Предисловие

Эта книга предназначена для широкого круга специалистов, включая разработчиков, архитекторов, администраторов баз данных, инженеров DevOps и других читателей, заинтересованных в эффективном изучении Elasticsearch и применении его¹ в своих приложениях — как новых, так и уже существующих. Особенно она полезна для тех, кто хочет работать с данными с помощью Elasticsearch.

Для чтения книги желательно иметь базовые знания computer science, а также быть знакомыми с JSON и REST. В книге рассматривается Elasticsearch, а также введение в другие инструменты стека Elastic.

Никаких знаний Elasticsearch не требуется, поскольку структура книги предполагает изучение основ и постепенный переход к продвинутым темам в понятной форме и с примерами практического применения. Благодаря такому подходу любой читатель сможет освоить представленные в ней концепции.

К концу книги вы будете хорошо разбираться в системе Elasticsearch и сможете использовать ее для управления и анализа огромных объемов данных. Книга представляет собой исчерпывающее руководство по управлению данными с помощью Elasticsearch, что делает ее незаменимой для разработчиков, аналитиков данных и всех тех, кто работает с данными. Я надеюсь, что она окажется информативной и полезной для вас.

Глава 1 «Начало работы с Elasticsearch» содержит обзор системы Elasticsearch и ее возможностей. В этой главе вы узнаете, что такое поиск и аналитика и почему они важны в современном мире, основанном на данных. Далее будет рассказано о том, как работает Elasticsearch, рассмотрены различные сценарии ее использования и ее преимущества. В заключение будет приведена краткая история Elasticsearch и ее эволюции с течением времени. К концу главы читатели будут хорошо понимать, что такое система Elasticsearch, ее назначение и то, как она вписывается в широкий контекст аналитики данных.

¹ Обычно Elasticsearch воспринимается и используется как «он» (поисковый движок, инструмент), если перед ним нет явного указания «система» или «экосистема», — в этом случае — «она». — Примеч. науч. ред.

Глава 2 «Установка Elasticsearch» содержит исчерпывающее руководство по установке и настройке Elasticsearch на различных операционных системах. Рассматриваются предварительные требования, необходимые для установки Elasticsearch, методы установки, а также способы настройки Elasticsearch для достижения оптимальной производительности. Кроме того, глава содержит инструкции по проверке установки и настройки. В результате у читателя окажется система Elasticsearch, готовая к работе на выбранной платформе.

Глава 3 «Elastic Stack: экосистема Elasticsearch» представляет обзор стека Elastic Stack, включающий инструменты Kibana, Logstash и Beats. В главе объясняется, как эти компоненты работают вместе, составляя полное решение для анализа данных. Читатели узнают о роли каждого компонента и о том, как их можно использовать для создания мощных дашбордов, визуализации данных и получения данных из различных источников.

Глава 4 «Подготовка данных к индексированию» рассматривает этапы подготовки данных к индексированию. В их число входит изучение типов анализаторов, нормализаторов, токенизаторов, фильтров токенов и фильтров символов, которые могут быть использованы в Elasticsearch для предварительной обработки данных. Мы рассмотрим практические примеры применения этих методов к различным источникам данных, таким как текстовые файлы, веб-логи и структурированные данные из баз данных. Вы получите четкое представление об оптимизации данных для эффективного и результативного индексирования в Elasticsearch.

Глава 5 «Импорт данных в Elasticsearch» расскажет, как импортировать в Elasticsearch данные из разных источников, таких как реляционные базы данных, CSV-файлы и т. д. Вы также узнаете, как преобразовывать и предварительно обрабатывать данные с помощью Logstash и Beats, двух важных компонентов стека Elastic. Кроме того, вы узнаете, как обрабатывать ошибки и контролировать процесс ввода данных для их эффективной обработки.

Глава 6 «Управление индексами: создание, обновление и удаление индексов Elasticsearch» раскрывает фундаментальные вопросы управления индексами Elasticsearch, включая их создание, обновление и удаление. В главе описываются особенности разных типов данных и маппинга, а также способы определения и управления элементами. Кроме того, рассматриваются методы обслуживания индексов, включая настройку распределения шардов и реализацию политик управления жизненным циклом индексов.

Глава 7 «Возможности поиска: освоение Query DSL и техник поиска» повествует о возможностях поиска в Elasticsearch. Вы поймете, что такое Query DSL и различные техники поиска, которые предлагает Elasticsearch. Вы научитесь писать сложные запросы для получения конкретной информации из

проиндексированных данных. Кроме того, вы узнаете о таких методах оптимизации поиска, как разбивка на страницы, сортировка и выделение.

Глава 8 «Работа с геоданными в Elasticsearch» посвящена использованию Elasticsearch для геопространственного поиска и анализа. В ней рассматриваются типы геопространственных данных, геозапросы, фильтрация по местоположению, геопространственные агрегации и геопространственный маппинг.

Глава 9 «Анализ данных с помощью агрегаций Elasticsearch» учит использовать агрегации Elasticsearch для анализа и обобщения данных. Агрегации позволяют выполнять сложные вычисления и получать информацию из данных, например находить среднее, максимальное, минимальное или наиболее часто встречающееся значение для определенного поля. Мы рассмотрим различные виды агрегаций, такие как агрегации метрик, бакетов и пайплайнов, а также способы их комбинирования для еще более эффективного анализа.

Глава 10 «Настройка производительности» рассказывает об оптимизации производительности Elasticsearch для работы с большими объемами данных. Мы рассмотрим аппаратные и сетевые параметры, управление памятью, распределение шардов и производительность индексирования. Вы также узнаете об инструментах и методах мониторинга и диагностики проблем производительности и о том, как настроить Elasticsearch для горизонтального масштабирования.

Глава 11 «Администрирование: управление кластерами Elasticsearch» посвящена администрированию кластеров Elasticsearch, включая управление и масштабирование. Будут рассмотрены такие темы, как управление кластером, управление узлами, распределение шардов и масштабирование кластеров Elasticsearch. Вы также узнаете о стратегиях резервного копирования и восстановления, функциях безопасности и мониторинге кластеров Elasticsearch.

От издательства

Мы выражаем огромную благодарность компании КРОК за помощь в работе над русскоязычным изданием книги и вклад в повышение качества переводной литературы.

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

О НАУЧНОМ РЕДАКТОРЕ РУССКОГО ИЗДАНИЯ

Анна Белых — старший инженер-разработчик в компании КРОК. Занимается проектированием и разработкой высоконагруженных информационных систем разных масштабов. Специализируется на вопросах производительности и оптимизации серверной (backend) части.

ГЛАВА 1

Начало работы с Elasticsearch

ВВЕДЕНИЕ

В этой главе мы поближе познакомимся с Elasticsearch. Начнем с обсуждения преимуществ Elasticsearch и того, как эта система может помочь компаниям достичь целей в области управления данными. Затем рассмотрим, что представляет собой Elasticsearch и как она использует мощный поисковый движок Lucene для быстрого и масштабируемого поиска. Чтобы как следует разобраться в основах Elasticsearch, мы рассмотрим некоторые базовые понятия, такие как узлы, кластеры, документы, индексы и шарды. Это необходимо для понимания того, как Elasticsearch хранит и организует данные в целях эффективного поиска и извлечения информации.

Затем мы рассмотрим некоторые **основные** сценарии использования Elasticsearch, включая поиск данных, ведение логов и анализ, мониторинг производительности приложений и систем, а также визуализацию данных. Эти примеры подчеркивают универсальность Elasticsearch и демонстрируют его аналитический потенциал в **широком** спектре отраслей и **приложений**. Кроме того, мы обсудим клиенты Elasticsearch, доступные для разработчиков, такие как Java, PHP, Perl, Python, .NET и JavaScript. Эти клиенты позволяют разработчикам применять поисковые возможности Elasticsearch в языке программирования, который они используют, и его экосистеме.

Наконец, мы обсудим, как использовать Elasticsearch в качестве первичного источника данных, вторичного источника данных или отдельной системы. Мы дадим рекомендации по принятию обоснованных решений о включении Elasticsearch в архитектуру данных на основе конкретных бизнес-потребностей и технических требований.

СТРУКТУРА ГЛАВЫ

В этой главе:

- Введение в поиск данных
- Что такое Elasticsearch и почему он важен для поиска и аналитики
- Обзор архитектуры и компонентов Elasticsearch
- Области применения и сценарии использования Elasticsearch
- Клиенты Elasticsearch и сценарии их использования

ЦЕЛИ

В этой главе представлен обзор движка Elasticsearch и его возможностей. В начале главы вводятся понятия поиска и аналитики и объясняется, почему они важны в современном мире, основанном на данных. Далее рассказывается о том, как эффективно использовать клиенты Elasticsearch.

ВВЕДЕНИЕ В ПОИСК ДАННЫХ

В современном мире экспоненциальный рост объема оцифрованных данных из различных источников, таких как смарт-устройства, датчики IoT и онлайн-транзакции, представляет собой серьезную проблему. Один из основных вызовов, стоящих перед отраслью, — преобразование неструктурированных данных в структурированную форму для оптимизации процесса хранения данных. Однако настоящая проблема кроется в поиске нужной информации в хранящихся данных. Традиционные системы хранения данных, такие как РСУБД, не подходят для текстового поиска из-за сложности написания SQL-запросов и неэффективности поиска даже после применения всех необходимых индексов. В отличие от них Elasticsearch, поисковая система на базе Lucene, предлагает сложный механизм поиска с выдачей релевантных результатов, агрегацией данных и многими другими преимуществами, недоступными РСУБД. Поэтому понимание важности поиска и того, как Elasticsearch способна помочь оптимизировать хранение и поиск данных, крайне важно для любой организации, работающей с большими объемами данных.

Поиск — важнейший компонент современных приложений, поскольку именно благодаря ему пользователи могут быстро и точно находить нужную информацию. Для всех приложений, работающих с большими объемами данных, будь то блог, интернет-магазин или любое другое приложение, механизм поиска необходим, чтобы предоставлять пользователям релевантные результаты.

Важность быстрых и точных результатов поиска трудно переоценить, поскольку пользователи, скорее всего, откажутся от приложения, которое не соответствует их требованиям к поиску. Поэтому оптимизация работы поиска чрезвычайно важна для эффективного взаимодействия с пользователями и сохранения их вовлеченности.

Помимо предоставления быстрых результатов, необходимо учитывать и другие важные параметры, такие как релевантность, агрегация и анализ данных. Это требование успешно обеспечивает Elasticsearch — мощная и масштабируемая поисковая система, способная работать с разными типами и источниками данных. Используя возможности Elasticsearch, приложения предоставляют пользователям быстрые, точные и релевантные результаты; это делает ее важнейшим компонентом современных приложений, ориентированных на поиск.

Важность функциональности поиска трудно переоценить. Помимо обеспечения быстрого отклика на запрос и получения релевантных результатов, существуют и другие критерии.

- **Предложение поиска.** Эффективная система поиска должна предлагать потенциальные поисковые запросы, как только пользователь начинает вводить текст.
- **Нечеткий поиск.** Система должна предлагать релевантные результаты, даже если пользователь допустил орфографическую ошибку в поисковом термине или использовал синоним.
- **Поиск по производным.** Качественная поисковая система должна распознавать производные поисковых терминов, например слова во множественном или единственном числе, чтобы предоставлять наиболее полные результаты.
- **Агрегация данных.** Система должна поддерживать агрегацию данных, чтобы предлагать пользователю дополнительные опции и фильтры, например ценовой диапазон, рейтинги, бренды и другую соответствующую информацию.
- **Релевантные результаты.** Результаты поиска должны выводиться в порядке соответствия запросу, с учетом таких факторов, как частота поискового термина, периодичность и поведение пользователей.
- **Расширенные фильтры.** Пользователи должны иметь возможность применять расширенные фильтры к результатам поиска, например по разрешению экрана, объему оперативной памяти, цвету и другим важным критериям.
- **Быстрое время отклика.** Поисковая система должна предоставлять результаты поиска быстро, в течение нескольких секунд, чтобы избежать

промедлений в работе пользователей и не вызывать у них негативных эмоций.

Учитывая эти критерии, разработчики могут создавать эффективные поисковые системы, выдающие быстрые и точные результаты, что в конечном итоге обеспечивает повышение вовлеченности и удовлетворенности пользователей.

ЧТО ТАКОЕ ELASTICSEARCH И ПОЧЕМУ ОН ВАЖЕН ДЛЯ ПОИСКА И АНАЛИТИКИ

Поисковый движок Elasticsearch был создан *Шаем Баноном (Shay Banon)*, основателем компании Elastic, которая занимается дальнейшей разработкой и поддержкой этого продукта. Elasticsearch – это ПО с открытым исходным кодом, которое может быть запущено на одном сервере или на сотнях серверов, чтобы обрабатывать петабайты данных. Elasticsearch – это мощная система, которая используется для поиска нужных данных в хранилищах большого объема.

В наш информационный век объем данных растет в геометрической прогрессии благодаря оцифровке и появлению новых источников данных, таких как смарт-устройства, датчики IoT (интернета вещей) и онлайн-транзакции. Эти данные могут быть структуризованными или неструктуризованными, относящимися к конкретному устройству или времененным рядам и поступать из разных источников, что затрудняет их поиск вручную. Для решения этих проблем Elasticsearch предоставляет распределенную, масштабируемую и документоориентированную поисковую систему на базе библиотеки Lucene. Lucene – это высокопроизводительная библиотека, которая обеспечивает быстрые и эффективные результаты поиска. Однако для ее использования требуется сложный код Java, и ее нелегко распределить по нескольким узлам.

Elasticsearch инкапсулирует все сложности Lucene и предоставляет REST API, которые облегчают пользовательское взаимодействие с Elasticsearch. Он также обеспечивает поддержку нескольких языков программирования с помощью языковых клиентов, поэтому можно писать код на желаемом языке и при этом взаимодействовать с Elasticsearch. Кроме того, с Elasticsearch можно взаимодействовать с помощью инструмента командной строки cURL.

Итак, Elasticsearch – это мощный поисковый и аналитический движок, обеспечивающий быстрый и эффективный поиск в больших объемах данных, что делает его жизненно важным инструментом для организаций, стремящихся извлечь из своих данных максимальную ценность.

ОБЗОР АРХИТЕКТУРЫ И КОМПОНЕНТОВ ELASTICSEARCH

Elasticsearch имеет распределенную архитектуру, которая позволяет обрабатывать большие объемы данных на нескольких узлах. Она включает несколько компонентов, которые работают вместе и обеспечивают масштабируемую и высокодоступную платформу для поиска и аналитики.

Узел

В Elasticsearch под узлом (node) понимается отдельный рабочий экземпляр поисковой системы. Elasticsearch состоит из одного или нескольких узлов, которые являются экземплярами сервера Elasticsearch. Например, в кластере из десяти серверов, на которых работает Elasticsearch, каждый сервер будет считаться узлом. В некоторых случаях для нерабочих сред может быть достаточно кластера с одним узлом. Однако по мере увеличения объема данных возникает необходимость в дополнительных узлах для горизонтального масштабирования кластера, что также обеспечивает отказоустойчивость. Зная о других узлах кластера, узел может передавать запросы клиентов соответствующему узлу. Стоит отметить, что узлы могут играть разные роли: узлы данных хранят и выполняют запросы; главные узлы управляют операциями всего кластера; узлы-координаторы пересыпают запросы на соответствующие узлы. Каждый узел работает независимо и взаимодействует с другими, образуя кластер. Узлы можно добавлять или удалять из кластера динамически, не влияя на работу всей системы. Узлы могут быть разных типов.

Узел, допустимый для выбора в качестве главного (**master-eligible node**)

В Elasticsearch 8 узел, допустимый для выбора в качестве главного узла, или мастера, отвечает за управление состоянием кластера, включая добавление или удаление узлов, распределение шардов между узлами и поддержание работоспособности кластера. Рекомендуется иметь в кластере не менее трех таких узлов-кандидатов, чтобы обеспечить высокую доступность и избежать ситуаций разделения мозга (split-brain) (см. врезку ниже).

Выделенный узел, допустимый для выбора в качестве главного (**dedicated master-eligible node**)

Выделенный узел, который является кандидатом в мастер-узел, — это узел в кластере Elasticsearch, который сконфигурирован как главный и не имеет никаких других обязанностей, таких как хранение данных или обработка поисковых

запросов. Задача выделенного узла — повысить стабильность и надежность кластера, в котором задачи главного узла выполняет выделенный узел.

Чтобы сконфигурировать узел как главный (мастер-узел) в Elasticsearch 8, необходимо задать следующие параметры в файле конфигурации `elasticsearch.yml`:

```
node.roles: [ master ]
```

Параметр `node.roles` должен иметь значение `master`, указывая, что этот узел может быть использован в качестве главного.

СИТУАЦИЯ РАЗДЕЛЕНИЯ МОЗГА (SPLIT-BRAIN)

В Elasticsearch ситуация разделения мозга возникает, когда кластер разделяется на несколько подкластеров, каждый из которых считает, что только он контролирует ситуацию. Это может произойти из-за нарушения связи между узлами в кластере вследствие сетевых проблем или других сбоев. В таких случаях каждый подкластер продолжает работать независимо, самостоятельно обновляя одни и те же данные, что может привести к несогласованности данных.

Ситуации split-brain можно предотвратить, создав систему на основе кворума, когда большинство узлов должно согласиться с принятым решением. В Elasticsearch это достигается путем наделения определенных узлов правами главных и требованием, чтобы для нормального функционирования кластера было доступно большинство таких узлов. Когда узел с правами главного обнаруживает, что больше не имеет связи с большинством этих узлов, он отключается и инициирует процесс выбора нового главного узла.

Важно правильно настроить Elasticsearch для обработки ситуаций разделения мозга, чтобы обеспечить согласованность данных и избежать их потери. Рекомендуется использовать выделенный главный узел и следить, чтобы в кластере всегда было нечетное количество узлов-кандидатов в главный узел (для гарантии большинства). Правильная настройка сетевых параметров и мониторинг потенциальных проблем также помогут предотвратить возникновение ситуаций split-brain.

ПРИМЕЧАНИЕ Важно отметить, что главный узел не должен быть перегружен другими задачами, поскольку ему требуется достаточно ресурсов для управления состоянием кластера. Если один главный узел выходит из строя или становится недоступным, оставшиеся узлы-кандидаты выбирают новый главный узел для управления состоянием кластера.

Голосующий узел (voting-only master-eligible node)

В Elasticsearch голосующий узел — это узел, который только участвует в процессе выбора главного узла, но сам не может им стать. Когда главный узел выходит из строя или становится недоступным, оставшиеся узлы должны выбрать новый главный узел для поддержания стабильности кластера. В это время узлы-кандидаты участвуют в выборе нового главного узла. Чтобы настроить в Elasticsearch 8 голосующий узел, необходимо задать следующие параметры в файле конфигурации `elasticsearch.yml`:

```
node.roles: [ data, master, voting_only ]
```

В примере выше мы задаем голосующий узел данных. Также можно задать выделенный голосующий узел, используя опцию в файле конфигурации `elasticsearch.yml`:

```
node.roles: [ master, voting_only ]
```

В примере выше мы устанавливаем голосующий узел, не имеющий обязанностей узла данных.

Узел данных (data node)

Узлы данных отвечают за хранение и управление данными, а также за выполнение CRUD-операций, поиска и агрегации данных. Можно настроить узел как узел данных, установив для параметра `node.data` значение `true` в файле конфигурации Elasticsearch. Если необходим отдельный узел данных, можно установить для других типов значение `false`, как показано в следующем фрагменте кода:

```
node.roles: [ data ]
```

В примере выше мы устанавливаем для параметра "node.role" значение `data`, а для всех остальных параметров — `false`, что делает узел выделенным узлом данных. Добавив в кластер больше узлов данных, можно горизонтально масштабировать кластер и обрабатывать большие объемы данных. Узлы данных также могут выполнять распределение и ребалансировку шардов, что помогает равномерно распределять данные по кластеру для повышения производительности и отказоустойчивости.

Узел поглощения данных (ingest node)

Узел поглощения данных — это специализированный тип узла в Elasticsearch, который позволяет выполнять предварительную обработку документов перед их индексацией. Он отвечает за обработку данных по мере их прохождения через пайплайн Elasticsearch, например, добавление в документы

дополнительных данных, обработку значений полей и выполнение преобразований данных.

Узлы поглощения имеют собственный специальный пайплайн, который можно настраивать для извлечения и преобразования данных с помощью различных процессоров, таких как grok, dissect и geoip. Это особенно полезно в случаях, когда необходимо извлечь релевантную информацию из неструктурированных данных, таких как файлы логов или потоки социальных сетей.

Чтобы сконфигурировать узел поглощения данных, установим для параметра `node.roles` в файле конфигурации Elasticsearch значение `ingest`. Вот пример:

```
node.roles: [ ingest ]
```

Узел поглощения позволяет выполнять предварительную обработку данных без необходимости писать собственный код или использовать внешние инструменты. Это упрощает пайплайн обработки данных и делает его более эффективным, особенно когда требуется обрабатывать большие объемы данных в реальном времени.

Узел машинного обучения (machine learning node)

Узел машинного обучения — это узел, способный выполнять задания машинного обучения на данных, хранящихся в кластере Elasticsearch. Узлы машинного обучения имеют специализированные аппаратные конфигурации и оптимизированы для обработки больших объемов данных в реальном времени.

Чтобы настроить узел машинного обучения, необходимо активировать функцию машинного обучения в файле конфигурации Elasticsearch и назначить узлу роль узла машинного обучения. Это можно сделать, добавив следующую строку в файл конфигурации `elasticsearch.yml`:

```
xpack.ml.enabled: true
```

Параметр `xpack.ml.enabled` включает функцию машинного обучения, а параметр `node.ml` назначает узлу роль узла машинного обучения.

После того как узел настроен, можно создавать задания машинного обучения с помощью API машинного обучения Elasticsearch. Эти задания могут анализировать данные в реальном времени и выявлять паттерны, аномалии и тенденции.

Чтобы создать выделенный узел машинного обучения, отредактируйте файл конфигурации Elasticsearch (`elasticsearch.yml`) и добавьте следующую строку:

```
node.roles: [ ml, remote_cluster_client ]
```

Эта строка указывает Elasticsearch настроить узел в качестве узла машинного обучения и узла удаленного клиента кластера. Настройка удаленного клиента кластера необходима, поскольку она позволяет узлу машинного обучения получать доступ к данным из других кластеров, что может понадобиться для анализа. Узел машинного обучения — платная функция Elasticsearch, которая позволяет запускать модели машинного обучения на данных Elasticsearch.

Предположим, что у вас есть узел машинного обучения в одном кластере Elasticsearch, но данные, которые вы хотите использовать для машинного обучения, хранятся в другом кластере. Настроив узел машинного обучения как узел удаленного клиента кластера, вы можете получить доступ к данным, хранящимся в другом кластере, не перенося их в кластер узла машинного обучения. Это позволяет сэкономить время и ресурсы, а также избежать ненужного дублирования данных.

Узел горячих данных (hot data node)

Узлы горячих данных — это особый тип узлов данных в Elasticsearch, которые оптимизированы для работы с высокопроизводительными рабочими нагрузками и высоким трафиком. Они предназначены для хранения наиболее часто используемых и запрашиваемых данных, также известных как *горячие данные*, и обычно развертываются на высокопроизводительном оборудовании для обеспечения быстрого времени отклика.

Узлы горячих данных характеризуются способностью обрабатывать большой объем запросов на чтение и запись в реальном времени. Они оптимизированы для эффективного индексирования и поиска данных, что делает их идеальными для таких сценариев использования, в которых требуется быстрый и частый доступ к данным, например для интернет-магазинов, социальных сетей и приложений финансовых услуг.

Чтобы настроить узел горячих данных, укажите следующие параметры в файле конфигурации Elasticsearch:

```
node.roles: [ data-hot ]
```

Используя настройки выше, можно определить узел горячих данных в Elasticsearch.

Узел теплых данных (warm data node)

Узел теплых данных в Elasticsearch — это тип узла, оптимизированный для хранения и поиска больших объемов данных, к которым обращаются реже. К ним могут относиться старые или реже используемые логи, исторические данные или резервные копии. Теплый узел обычно имеет меньшую производительность

хранения и меньшие требования к памяти по сравнению с горячими узлами, но он может хранить большой объем данных при меньших затратах.

Теплые узлы также предназначены для работы с нагрузками, связанными с чтением, и могут быть не так отзывчивы на запросы записи, как горячие узлы. Чтобы настроить узел теплых данных в Elasticsearch 8, задайте следующие параметры в файле конфигурации `elasticsearch.yml`:

```
node.roles: [ data-warm ]
```

Таким образом узел будет настроен для обработки теплых данных и оптимизирован для хранения нечасто используемых данных и доступа к ним.

Узел холодных данных (cold data node)

Узел холодных данных — это тип узла данных в Elasticsearch, специально предназначенный для хранения редко используемых или архивных данных. Холодные узлы обычно создаются более медленными и имеющими меньшие вычислительные ресурсы, что делает выгодным их использование для хранения больших объемов данных с редким доступом.

Чтобы настроить узел холодных данных в Elasticsearch 8, измените параметр `node.roles` в файле конфигурации `elasticsearch.yml` следующим образом:

```
node.roles: [ data-cold ]
```

Таким образом узел будет настроен для обработки холодных данных и оптимизирован для хранения редко используемых данных и управления ими, в то время как другие узлы в кластере будут обрабатывать более активные и часто используемые данные.

Узлы холодных данных полезны для долгосрочного хранения данных, а также соблюдения нормативных и регуляторных требований. Отделение холодных данных от горячих или теплых позволит оптимизировать распределение ресурсов и производительность кластера Elasticsearch. Узлы холодных данных обычно используются для данных, которые не требуют частых запросов или анализа в реальном времени, но при этом должны храниться и быть доступными для соблюдения нормативных требований или для исторических целей.

Узел замороженных данных (frozen data node)

В Elasticsearch 8 узел замороженных данных — это специализированный тип узла, оптимизированный для хранения данных, к которым обращаются редко или которые доступны только для чтения. Узлы замороженных данных предназначены для эффективного и малозатратного хранения больших

объемов данных, которые не запрашиваются и не обновляются. Такие узлы особенно хорошо подходят для долгосрочного архивирования и обеспечения соответствия нормативным требованиям, когда данные должны храниться в течение определенного времени, но доступ к ним осуществляется редко. Отделение замороженных данных от других типов, таких как горячие или теплые данные, позволяет оптимизировать использование ресурсов и повысить общую производительность кластера. Узлы замороженных данных позволяют хранить большие объемы данных и управлять ими с малыми затратами, обеспечивая при этом доступность данных и соответствие нормативным требованиям.

Чтобы настроить узел замороженных данных в Elasticsearch, измените параметр `node.roles` в файле конфигурации `elasticsearch.yml` следующим образом:

```
node.roles: [ data-frozen ]
```

Таким образом узел будет настроен для обработки замороженных данных и оптимизирован для их хранения и управления ими.

ПРИМЕЧАНИЕ Важно отметить, что узлы замороженных данных предназначены только для чтения, то есть данные в них можно записывать лишь на этапе начального индексирования. После того как данные будут проиндексированы и сохранены, их нельзя будет изменить или обновить. Однако они станут доступными для поиска и извлечения, что делает узлы замороженных данных полезным средством для хранения и архивирования крупных данных в Elasticsearch.

Кластер

В Elasticsearch кластер — это набор узлов, которые взаимодействуют между собой, создавая целостную и распределенную среду для хранения и обработки данных. Каждый кластер идентифицируется уникальным именем, что позволяет узлам соединяться и взаимодействовать с конкретным кластером, к которому они принадлежат. Узлы внутри кластера работают вместе, обеспечивая единое и согласованное представление данных. Они взаимодействуют друг с другом для равномерного распределения и репликации данных на нескольких узлах; это позволяет повысить доступность данных, отказоустойчивость и общую производительность системы.

Распределяя данные по узлам, кластер обеспечивает горизонтальную масштабируемость. По мере увеличения объема данных или роста рабочей нагрузки в кластер могут быть добавлены дополнительные узлы, чтобы он мог справиться с возросшими требованиями к хранению и обработке данных. Такая масштабируемая архитектура позволяет Elasticsearch обрабатывать

большие массивы данных и эффективно справляться с большими объемами запросов.

Кластеры также играют важную роль в обеспечении надежности и отказоустойчивости данных. Благодаря репликации данных на нескольких узлах кластер может выдерживать сбои и предотвращать потерю данных. Если один из узлов становится недоступным или выходит из строя, кластер автоматически перераспределяет хранящиеся на нем данные на другие доступные узлы, поддерживая необходимую избыточность данных и гарантируя их доступность даже при сбое узлов.

Помимо распределения данных и обеспечения отказоустойчивости, кластеры предоставляют централизованную точку управления для мониторинга и администрирования. Такие операции, как мониторинг состояния кластера, управление индексами и ребалансировка данных, могут выполняться на уровне кластера; таким образом осуществляется единый и систематизированный подход к управлению всем развертыванием Elasticsearch.

Индекс

В Elasticsearch индекс представляет собой логическое пространство имен, или контейнер, в котором хранится коллекция документов. Индекс можно рассматривать как традиционную базу данных, где данные организованы и хранятся в структурированном виде. Задача индекса — сгруппировать документы, которые имеют схожие характеристики или принадлежат к одной категории. Например, в приложении онлайн-магазина можно задать отдельные индексы для товаров, клиентов и заказов. Это позволит выполнять эффективный поиск и извлекать необходимую информацию по определенным предметным областям (доменам).

Elasticsearch использует структуру инвертированного индекса для быстрого полнотекстового поиска. Инвертированный индекс состоит из списка уникальных терминов, встречающихся во всех документах в индексе, а также идентификаторов документов, указывающих на вхождения каждого термина. Подобная техника индексирования значительно ускоряет операции поиска за счет предварительного вычисления связей между терминами и документами.

Для обработки больших объемов данных и распределения рабочей нагрузки индекс делится на один или несколько шардов. Каждый шард — это независимое подмножество данных индекса, которое может храниться на отдельном узле в кластере Elasticsearch. Разбивая индекс на шарды, Elasticsearch может распараллелить операции поиска и индексирования, повышая производительность и масштабируемость.

Шарды

В Elasticsearch индекс состоит из одного или нескольких шардов, и каждый шард является самостоятельной единицей индекса. Разбивая индекс на более мелкие шарды, Elasticsearch распределяет данные и операции между некоторыми узлами, что повышает производительность и масштабируемость системы. Шарды позволяют Elasticsearch распараллеливать операции поиска и индексирования. Выдаваемый запрос на поиск параллельно передается во все шарды, а результаты поиска объединяются и возвращаются пользователю. Такое распараллеливание позволяет Elasticsearch обрабатывать большие объемы данных и сложные поисковые запросы.

При создании индекса можно указать количество шардов, и Elasticsearch автоматически распределит их между доступными узлами кластера. Необходимое количество шардов индекса зависит от разных факторов, таких как размер индекса, количество документов и ожидаемая производительность поиска и индексирования. Elasticsearch также поддерживает возможность создания реплик шардов, которые представляют собой копии основных шардов. Шарды-реплики обеспечивают избыточность и высокую доступность, позволяя системе продолжать функционировать, даже если некоторые узлы выйдут из строя. При создании индекса можно также указать количество реплик. Реплики распределяются по доступным узлам кластера, отдельно от первичных шардов.

Например, если необходимо проиндексировать 100 Гбайт данных и настроены четыре шарда, то 100 Гбайт данных будут разделены на шарды по 25 Гбайт. Если узел один, все четыре шарда останутся на этом узле. Если добавить в кластер еще один узел, шарды будут равномерно распределены на обоих узлах. Так, два шарда останутся на узле 1, а два переместятся на второй узел кластера.

Шарды могут быть двух типов: первичные и реплики. Первичные шарды содержат первичные данные, а реплики — копии первичных шардов. Мы используем реплики, чтобы защититься от любых аппаратных сбоев и повысить производительность поиска в кластере.

На следующем рисунке (рис. 1.1) показан кластер Elasticsearch с тремя узлами. Конфигурация шардов на нем следующая:

- количество первичных шардов: 2;
- количество реплик шардов: 1.

Теперь, если узлов в кластере два — один первичный и одна реплика, — два шарда переместятся на один узел, а первичный второй шард и первая реплика шарда — на другой.

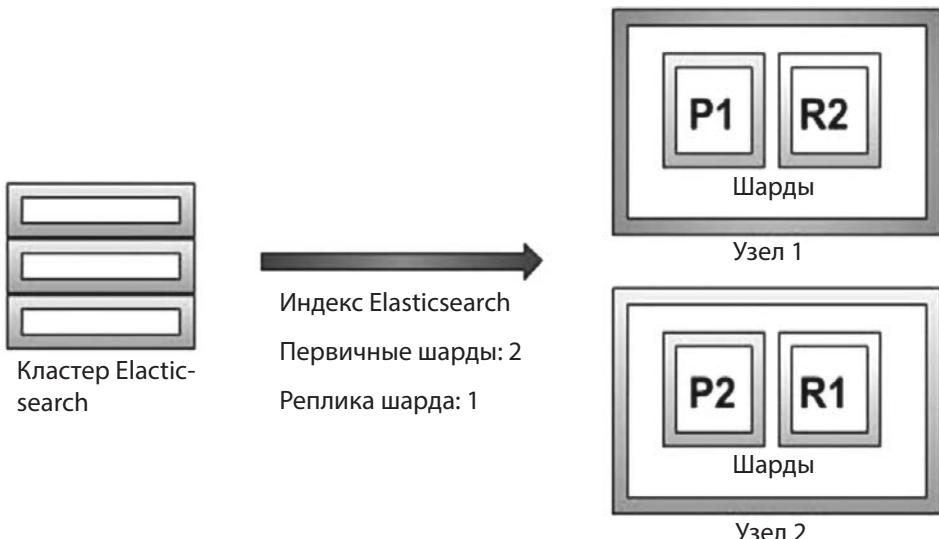


Рис. 1.1. Шарды Elasticsearch

На рисунке выше показан кластер с узлом 1 и узлом 2. На узле 1 находятся шарды P1 и R2, а на узле 2 – шарды P2 и R1. Р обозначает первичные шарды, а R – шарды-реплики.

Документы

В Elasticsearch документ – это основная единица информации, которую можно индексировать и искать. Он представляет собой один объект данных, обычно в формате JSON, и содержит фактические данные, которые хранятся и извлекаются в Elasticsearch. Документы в Elasticsearch организованы в индексе и однозначно идентифицируются по идентификатору. Каждый документ состоит из полей, которые представляют собой пары «ключ – значение», отражающие атрибуты или свойства данных. Типы данных полей могут быть различными, включая строки, числа, булевы выражения, даты и др.

Документы в Elasticsearch являются гибкими, то есть не требуют предопределенной схемы. Такая гибкость позволяет осуществлять динамическое индексирование, когда в документ можно добавлять новые поля, не изменяя структуру индекса. Она также позволяет работать с разными типами и структурами данных в рамках одного индекса.

Когда документ индексируется в Elasticsearch, он сохраняется и становится доступным для поиска в указанном индексе. Документ автоматически

назначается одному или нескольким шардам в зависимости от конфигурации индекса. Процесс индексирования включает в себя анализ текста и извлечение релевантных терминов для построения инвертированного индекса, обеспечивающего эффективный полнотекстовый поиск.

ОБЛАСТИ ПРИМЕНЕНИЯ И ВАРИАНТЫ ИСПОЛЬЗОВАНИЯ ELASTICSEARCH

Мы уже обсуждали некоторые возможности Elasticsearch, например аналитику, которая позволяет формировать срезы данных, чтобы извлечь максимально полную информацию. Аналитика позволяет искать неточные совпадения или, благодаря нечеткому поиску, выдавать результаты, даже если пользователь неверно ввел слово. Таким образом, функции Elasticsearch оказываются полезными в разных ситуациях. Перечислить все потенциальные сценарии использования Elasticsearch невозможно, назовем лишь основные.

Поиск данных

Основная задача Elasticsearch — поиск данных, особенно при работе с большими массивами данных. В прошлом для хранения и поиска данных обычно использовались реляционные БД. Однако в современных условиях реляционные базы данных часто не в состоянии обеспечить оптимальную производительность поиска, поскольку они не предназначены для работы в качестве поисковых систем.

А Elasticsearch является высокомасштабируемой поисковой системой, которая обеспечивает быстрый поиск. Ее возможности выходят за рамки базовых и включают такие функции, как агрегация, анализ и нечеткий поиск. Эти функции делают Elasticsearch идеальным инструментом для разнообразных задач поиска данных.

Онлайн-магазины, сайты продажи путевок, платформы социальных сетей, сайты биоинформатики, новостные порталы, блоги и многие другие ресурсы используют для поиска данных движок Elasticsearch. Чтобы оставаться конкурентоспособными на своих рынках, компании должны быстро выдавать результаты поиска. Любые задержки приводят к неудовлетворенности клиентов и потенциальным убыткам.

Таким образом, поиск данных остается основной областью применения Elasticsearch, и многие компании используют его возможности в этой области для улучшения приложений и повышения качества взаимодействия с пользователями.

Регистрация и анализ данных

Регистрация и анализ данных — еще одна важная область, в которой Elasticsearch активно используется совместно с другими инструментами из стека Elastic, такими как Beats, Logstash и Kibana. В этом контексте Beats и Logstash выступают как инструменты ввода данных, которые позволяют собирать данные из различных источников, включая файлы логов, и обеспечивают их беспрепятственную передачу в Elasticsearch. После успешного индексирования данных в Elasticsearch в качестве мощной платформы аналитики и визуализации данных логов используется Kibana.

Анализ данных позволяет организациям эффективно контролировать свои системы, выявлять потенциальные проблемы и проактивно реагировать на любые аномалии. Такой комплексный подход используется для отслеживания и расследования событий, устранения неполадок и получения аналитических данных о деятельности компании. Возможности Beats и Logstash позволяют собирать широкий спектр данных, включая данные логов и приложений, сетевые данные, а также системные метрики. Эти агрегированные данные в дальнейшем можно детально проанализировать и исследовать с помощью Elasticsearch и Kibana.

Многие компании используют стек Elastic Stack для централизованного анализа данных, что позволяет эффективно отслеживать производительность и поведение работающих приложений. Используя возможности Elasticsearch в сочетании с другими компонентами Elastic Stack, компании могут создавать надежную и масштабируемую инфраструктуру для регистрации, поглощения, анализа и визуализации данных, способствующую принятию информированных решений и эффективному устранению проблем.

Мониторинг производительности приложений

Application Performance Monitoring (APM) в Elastic Stack — это инструмент с открытым исходным кодом, предназначенный для мониторинга производительности приложений. Он состоит из сервера (APM Server) и агентов (APM Agents). Сервер APM выступает в качестве центрального хаба, получая данные от агентов APM и перенаправляя их в Elasticsearch для хранения и анализа. Агенты APM, с другой стороны, являются языковыми и сконфигурированы для языков, поддерживаемых Elastic APM. После настройки они начинают собирать и отправлять метрики приложений на APM-сервер.

Передавая собранные данные в Elasticsearch, разработчики и системные администраторы могут использовать возможности мониторинга Elastic APM, чтобы получать ценную аналитику производительности и доступности приложений. APM упрощает выявление и диагностику системных проблем, что позволяет

принимать проактивные меры для устранения потенциальных узких мест или аномалий. Одно из ключевых преимуществ Elastic APM – обеспечение видимости на уровне кода. Она позволяет разработчикам отслеживать и анализировать выполнение кода, предоставляя им ценные сведения для выявления и устранения проблем в коде. Благодаря функции поиска в Elasticsearch проблемы кода можно искать в APM, тем самым эффективно улучшая качество кода.

Мониторинг и анализ данных о производительности приложений можно осуществлять с помощью Kibana, которая предоставляет специальный пользовательский интерфейс для Elastic APM. Этот интерфейс дает возможность пользователям легко отслеживать метрики производительности приложений и создавать пользовательские дашборды на основе данных APM. Благодаря гибкости APM и возможностям визуализации Kibana разработчики и системные администраторы могут получать полное представление о поведении приложений, что позволит оптимизировать производительность, устранять проблемы и повышать общее качество приложений.

Мониторинг производительности системы

Мониторинг системы очень важен для поддержания оптимальной производительности приложения. На нее могут влиять различные факторы, включая загрузку процессора, использование памяти и производительность базы данных. Постоянный мониторинг этих метрик позволяет выявлять потенциальные проблемы и решать их проактивно, что улучшает пользовательский опыт.

Агенты Elastic Beats, такие как Metricbeat, Packetbeat и Heartbeat, можно настроить на сбор системных метрик, деталей сетевых пакетов и данных о времени работы сервисов/API с разных серверов. Собранные данные отправляются в Elasticsearch для хранения и анализа. Kibana может использоваться для визуализации и анализа системных метрик в целях анализа производительности базовой инфраструктуры. Мониторинг производительности системы – распространенный и важный вариант использования Elasticsearch. Он позволяет организациям следить за состоянием и производительностью инфраструктуры, обеспечивая своевременное обнаружение и решение потенциальных проблем.

Визуализация данных

Визуализация данных – один из важных сценариев использования Elasticsearch, позволяющий организациям получать информацию и принимать взвешенные решения на основе данных. Собирая данные из разных источников и храня их в Elasticsearch, организации могут использовать такие инструменты визуализации, как Kibana, Grafana или Graylog, для создания интерактивных и информативных дашбордов.

Эти инструменты визуализации позволяют строить графики и организовывать визуально привлекательное представление данных. Организации могут определять ключевые показатели эффективности (KPI) и создавать пользовательские визуализации для отображения тенденций, закономерностей и корреляций любых типов данных — числовых, категориальных или текстовых.

В случае использования Elasticsearch в качестве базового хранилища данных инструменты визуализации предлагают гибкие и мощные функции для изучения, анализа и представления данных. Это дает возможность пользователям получать практические выводы, контролировать ключевые метрики и отслеживать эффективность бизнес-операций.

Elasticsearch можно использовать в самых разных приложениях и сценариях для поиска и аналитики. Вот несколько примеров:

- **Электронная коммерция.** Elasticsearch часто используется в приложениях онлайн-магазинов для поиска товаров и рекомендаций. Он помогает покупателям найти товары по запросам и предоставляет предложения на основе истории поиска.
- **Анализ логов.** Elasticsearch можно использовать для анализа больших объемов данных логов в реальном времени. Он помогает быстро выявлять проблемы и ошибки, а также обеспечивает быстрый поиск и фильтрацию логов.
- **Корпоративный поиск.** Elasticsearch можно использовать для создания приложений корпоративного поиска, которые позволяют сотрудникам быстро искать информацию в разных источниках внутри организации.
- **Аналитика безопасности.** Elasticsearch можно использовать для анализа данных, связанных с безопасностью, таких как логины, сетевой трафик и оповещения. С его помощью можно выявить закономерности и аномалии, указывающие на потенциальные угрозы безопасности.
- **Анализ геопространственных данных.** Elasticsearch можно использовать для анализа геопространственных данных, таких как данные GPS, карты и информация о местоположении. С его помощью можно выявить тенденции и закономерности в данных, связанных с географическим положением.
- **Метрики и мониторинг производительности.** Elasticsearch можно использовать для мониторинга и анализа метрик и данных о производительности, таких как метрики сервера, метрики приложений и сетевые метрики.

В целом Elasticsearch — это универсальный инструмент, который можно применять в самых разных приложениях и сценариях, где важны поиск и аналитика. Существует также решение Elastic **Security Information and Event Management (SIEM)**, которое помогает организациям собирать, анализировать

и реагировать на угрозы безопасности. Оно построено на базе стека Elastic Stack и предоставляет функции сбора и анализа данных, реагирования на инциденты и создания отчетов.

КЛИЕНТЫ ELASTICSEARCH И СЦЕНАРИИ ИХ ИСПОЛЬЗОВАНИЯ

Elasticsearch предоставляет REST API для взаимодействия. Однако для его организации необходимо создать REST-клиент, написать JSON-запрос и выполнить его с помощью REST-клиента. Помимо написания запросов к Elasticsearch, можно использовать клиенты Elasticsearch для разных языков программирования:

- Java;
- PHP;
- Perl;
- Python;
- .NET;
- Ruby;
- JavaScript.

Используя языковые клиенты Elasticsearch, можно взаимодействовать с Elasticsearch на соответствующем языке. Это избавляет от необходимости изучать новые инструменты. Теперь рассмотрим, как работать с Elasticsearch с помощью этих клиентов. Представим их кратко, не вдаваясь в подробности.

Java

Java-клиенты бывают двух типов:

- низкоуровневый REST-клиент;
- высокоуровневый REST-клиент.

Низкоуровневые клиенты используются для взаимодействия через HTTP с кластером Elasticsearch. А высокоуровневый REST-клиент создается на основе низкоуровневого и раскрывает API. Такие клиенты отправляют запрос, преобразуя данные в поток байтов, — этот процесс называется маршалингом (*marshalling*). Ответ отображается пользователю путем преобразования потока байтов в данные — такой процесс называется демаршалингом (*unmarshalling*).

PHP

PHP-клиент Elasticsearch — это низкоуровневый клиент, который работает как любой другой REST API. В PHP-клиенте создаются ассоциативные массивы для разнообразных операций, таких как индексация, поиск и т. д. Например, если требуется проиндексировать документ после создания клиента, необходимо выполнить следующий фрагмент кода:

```
1. $index_data = [
2.   'index' => 'index_name',
3.   'id' => 'id',
4.   'body' => ['field_name' => 'field_value']
5. ];
6. $response = $client->index($index_data);
```

Не волнуйтесь, если вы не понимаете код; он предназначен для пользователей, которые знают PHP.

Perl

В Elasticsearch есть клиент `Search::Elasticsearch`, с помощью которого можно взаимодействовать с Elasticsearch из программы на Perl. После создания клиента Elasticsearch можно проиндексировать документ с помощью следующего фрагмента кода:

```
1. $client-> index(
2.   index => 'index_name',
3.   id => 'id',
4.   body => {
5.     'field_name' => 'field_value'
6.   }
7. );
```

Python

Клиент Elasticsearch Python транслирует тип данных Python в JSON и обратно, поскольку Elasticsearch принимает данные JSON. Разработчики Python могут писать код на Python и взаимодействовать с Elasticsearch. Для индексирования документа с помощью клиента Python необходимо выполнить следующий фрагмент кода:

```
1. es.index(index = "my-index", id = "id", body = {
2.   'field_name' : 'field_value'
3. })
```

.NET

Существует два клиента .NET для Elasticsearch: Elasticsearch.Net и NEST. Elasticsearch.Net – это низкоуровневый клиент, а NEST – высокоуровневый. Для индексирования документа с помощью клиента .NET необходимо выполнить фрагмент кода, приведенный ниже.

В примере мы создаем объект РОКО сотрудника (Employee):

```
1. var Employee = new Employee
2. {
3.     Id = 1,
4.     FirstName = "Sophia",
5.     LastName = "Julian"
6. };
7.
8. var indexResponse = client.IndexDocument(employee);
```

Ruby

Для взаимодействия с Elasticsearch на Ruby существует клиент RubyGem. Для индексирования документа с его помощью необходимо выполнить следующий фрагмент кода:

```
client.index index: 'index_name', id: 'id', body:
{ field_name: 'field_value' }
```

JavaScript

Elasticsearch предоставляет официальный клиент Node.js для JavaScript. Для индексирования документа с его помощью необходимо выполнить такой фрагмент кода:

```
1. await client.index({
2.   index: 'index_name',
3.   body: {
4.     field_name: 'field_value'
5.   }
6. })
```

Таким образом можно настроить клиент и с его помощью взаимодействовать с Elasticsearch с целью индексации документов, поиска в них и т. д. Мы рассмотрели некоторые из клиентов. Есть и другие, но обсуждение каждого из них не входит в задачи книги.

ЗАКЛЮЧЕНИЕ

В этой главе мы познакомились с Elasticsearch и его основными концепциями. Мы подчеркнули важность поиска и рассмотрели сценарии использования, включая поиск, регистрацию и анализ данных, мониторинг производительности приложений, мониторинг производительности системы и визуализацию данных. Мы также обсудили клиенты Elasticsearch для бесшовной интеграции. Разобравшись в этих основах, вы теперь готовы использовать возможности Elasticsearch в своих проектах.

В следующей главе вы изучите порядок установки Elasticsearch на различных платформах и REST API, которые он предоставляет.

ВОПРОСЫ

1. Что такое Elasticsearch и где его можно использовать?
2. Сколько типов узлов существует в Elasticsearch?
3. Что такое клиенты Elasticsearch и какие языки они поддерживают?
4. Выберите языковой клиент Elasticsearch и попробуйте использовать его для своего языка программирования.
5. Расскажите о разных сценариях использования Elasticsearch.

ГЛАВА 2

Установка Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы говорили о важности поиска данных и разбирали варианты его применения. Мы также рассмотрели основы Elasticsearch, включая его компоненты, сценарии использования и поддерживаемые клиенты. Теперь посмотрим, что нового появилось в Elasticsearch 8. Займемся установкой Elasticsearch 8.7 на разные платформы, следя за корректностью настройки. После установки мы погрузимся в многообразие REST API, предоставляемых Elasticsearch и позволяющих эффективно взаимодействовать с системой и использовать ее многочисленные возможности.

СТРУКТУРА

В этой главе:

- Введение в Elasticsearch 8
- Установка Elasticsearch на Linux и macOS
- Установка Elasticsearch с помощью пакета Debian
- Установка Elasticsearch с помощью пакета RPM
- Установка Elasticsearch на Windows
- Запуск и проверка службы Elasticsearch
- REST-интерфейсы Elasticsearch
- API кластеров

ЦЕЛИ

По завершении этой главы вы получите полное представление об Elasticsearch 8 и его ключевых возможностях. Вы научитесь устанавливать Elasticsearch в своей системе, что позволит вам использовать его в соответствии с вашими требованиями. Кроме того, вы узнаете, как выполнять REST-запросы к Elasticsearch, чтобы взаимодействовать с кластером Elasticsearch и осуществлять различные операции, такие как индексирование, поиск и анализ данных. В целом эта глава закладывает основы, необходимые для эффективного управления и поиска данных с помощью Elasticsearch 8.

ВВЕДЕНИЕ В ELASTICSEARCH 8

В 8-й версии Elasticsearch реализован ряд заметных улучшений и усовершенствований общей производительности и функциональности по сравнению с 7-й. Подробно рассказать обо всех невозможно, поэтому выделим основные.

Повышение производительности индексирования

В Elasticsearch 8 реализован более эффективный и быстрый механизм индексирования, благодаря чему повышена пропускная способность индексирования и уменьшена задержка. Это усовершенствование позволяет быстрее вводить данные в Elasticsearch, обеспечивая выполнение сценариев индексирования в реальном времени.

Повышение производительности поиска

В Elasticsearch 8 реализованы различные оптимизации для повышения производительности поиска. Они включают в себя более эффективное управление памятью, а также улучшенные стратегии кэширования и выполнения запросов для ускорения и повышения эффективности поиска.

Улучшения межклusterного поиска

В Elasticsearch 8 улучшена функциональность межклusterного поиска, обеспечивающая эффективный поиск по нескольким кластерам. Это улучшение повышает масштабируемость и гибкость распределенных сред с несколькими кластерами Elasticsearch.

Усовершенствования в области безопасности

В Elasticsearch 8 представлены расширенные функции безопасности, включая усовершенствованные механизмы аутентификации и авторизации, улучшенный **контроль доступа на основе ролей (RBAC)** и более строгие конфигурации безопасности. Эти усовершенствования повышают безопасность развертываний Elasticsearch, защищая конфиденциальные данные и предотвращая несанкционированный доступ.

Усовершенствования в области эксплуатации

Elasticsearch 8 включает в себя различные улучшения, касающиеся эксплуатации, такие как более эффективные стратегии совместного распределения, более эффективная обработка изменений состояния кластера и улучшенные API управления кластером. Все они упрощают управление и эксплуатацию кластеров Elasticsearch, повышая общую стабильность и отказоустойчивость системы.

Важно отметить, что это лишь несколько примеров. Elasticsearch 8 содержит целый ряд улучшений в разных областях системы для более высокой производительности, масштабируемости, безопасности и расширенных возможностей эксплуатации.

Прежде чем переходить к API Elasticsearch и другим деталям, необходимо понять, как установить Elasticsearch на разных платформах. Рассмотрим этот процесс.

УСТАНОВКА ELASTICSEARCH В LINUX И MACOS

Elasticsearch предоставляет архив **.tar.gz** для Linux и macOS, распространяемый бесплатно в соответствии с лицензией Elastic. Он содержит как открытый исходный код, так и коммерческие функции. Для использования коммерческих функций необходимо приобрести лицензию. Доступна также 30-дневная пробная версия. Последнюю стабильную версию Elasticsearch можно найти на странице загрузки Download Elasticsearch на сайте Elastic. Рассмотрим Elasticsearch версии 8.7.1.

Установка Elasticsearch в Linux

Чтобы установить Elasticsearch в Linux, необходимо проделать шаги, указанные ниже.

1. Загрузить tar-файл Elasticsearch 8.7.1 Linux. Для этого выполнить команду

```
wget https://artifacts.elastic.co/downloads/elasticsearch/  
elasticsearch-8.7.1-linux-x86_64.tar.gz
```

2. После загрузки архива сравнить **Secure Hash Algorithm (SHA)** с опубликованной контрольной суммой. Для сравнения необходимо выполнить команды:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/  
elasticsearch-8.7.1-linux-x86_64.tar.gz.sha512  
shasum -a 512 -c elasticsearch-8.7.1-linux-x86_64.tar.gz.sha512
```

3. После загрузки и сравнения контрольной суммы извлечь архив с помощью команды

```
tar -xzf elasticsearch-8.7.1-linux-x86_64.tar.gz  
cd elasticsearch-8.7.1/
```

4. После извлечения архива структура каталогов становится видимой в домашней директории Elasticsearch. Теперь необходимо запустить службу Elasticsearch, выполнив следующую команду:

```
./bin/elasticsearch
```

Установка Elasticsearch в macOS

Чтобы установить Elasticsearch в macOS, необходимо выполнить следующие шаги.

1. Загрузить tar-файл Elasticsearch 8.7.1 macOS с помощью команды

```
curl -O https://artifacts.elastic.co/downloads/elasticsearch/  
elasticsearch-8.7.1-darwin-x86_64.tar.gz
```

2. Сравнить SHA с опубликованной контрольной суммой. Для сравнения необходимо выполнить следующие команды:

```
curl https://artifacts.elastic.co/downloads/elasticsearch/  
elasticsearch-8.7.1-darwin-x86_64.tar.gz.sha512 | shasum -a 512 -c -.
```

3. После загрузки и сравнения контрольной суммы извлечь архив с помощью команды

```
tar -xzf elasticsearch-8.7.1-darwin-x86_64.tar.gz  
cd elasticsearch-8.7.1/
```

4. После извлечения архива структура каталогов становится видимой в домашней директории Elasticsearch. Теперь необходимо запустить службу Elasticsearch, выполнив следующую команду:

```
./bin/elasticsearch
```

Теперь рассмотрим, как установить Elasticsearch с помощью пакета Debian.

УСТАНОВКА ELASTICSEARCH С ПОМОЩЬЮ ПАКЕТА DEBIAN

В системах на базе Debian, таких как Debian и Ubuntu, установить Elasticsearch можно с помощью пакета Debian для Elasticsearch. Он доступен для скачивания в разделе загрузки на сайте Elastic. Этот пакет бесплатный, а тестировать коммерческие возможности можно в течение 30 дней после установки Elasticsearch, если выбрать пробную версию. Чтобы установить Elasticsearch с помощью пакета Debian, необходимо выполнить следующие шаги.

1. Установить открытый ключ подписи после его загрузки, используя команду

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo gpg --dearmor -o /usr/share/keyrings/elasticsearch-keyring.gpg
```

2. После установки открытого ключа подписи установить пакет `apt-transport-https` с помощью команды

```
sudo apt-get install apt-transport-https
```

3. Сохранить определение репозитория в файле `/etc/apt/sources.list.d/elastic-8.x.list`. Для этого выполнить команду

```
echo "deb [signed-by=/usr/share/keyrings/elasticsearch-keyring.gpg] https://artifacts.elastic.co/packages/8.x/apt stable main" | sudo tee /etc/apt/sources.list.d/elastic-8.x.list
```

4. Установить пакет Elasticsearch Debian с помощью команды

```
sudo apt-get update && sudo apt-get install elasticsearch
```

Установка пакета Debian вручную

Загрузить и установить пакет Elasticsearch в Debian можно вручную, выполнив следующие шаги.

1. Загрузить файл пакета Elasticsearch 8.7.1 Debian с помощью команды

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-8.7.1-amd64.deb
```

2. После загрузки пакета Debian сравнить SHA с опубликованной контрольной суммой. Для сравнения необходимо выполнить команды

```
wget https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-8.7.1-amd64.deb.sha512
shasum -a 512 -c elasticsearch-8.7.1-amd64.deb.sha512
```

3. После загрузки и сравнения контрольной суммы извлечь архив с помощью команды

```
sudo dpkg -i elasticsearch-8.7.1-amd64.deb
```

УСТАНОВКА ELASTICSEARCH С ПОМОЩЬЮ ПАКЕТА RPM

На системах с RPM, таких как openSUSE, SLES, Centos, Red Hat и Oracle Enterprise, можно установить Elasticsearch с помощью пакета RPM. Он доступен для скачивания в разделе загрузки на сайте Elastic. Пакет RPM также является бесплатным, и тестировать коммерческие функции можно в течение 30 дней после установки Elasticsearch. Чтобы установить Elasticsearch с помощью пакета RPM, необходимо выполнить следующие шаги.

1. Установить открытый ключ подписи Elasticsearch после его загрузки. Это делает команда

```
rpm --import https://artifacts.elastic.co/GPG-KEY-elasticsearch
```

2. Создать файл `elasticsearch.repo` в каталоге `/etc/yum.repos.d/` для дистрибутивов на базе RedHat. Этот файл должен содержать следующее:

```
[elasticsearch].  
name=Elasticsearch repository for 8.x packages  
baseurl=https://artifacts.elastic.co/packages/8.x/yum  
gpgcheck=1  
gpgkey=https://artifacts.elastic.co/GPG-KEY-elasticsearch  
enabled=0  
autorefresh=1  
type=rpm-md
```

3. Сохранить файл. Теперь все готово к установке Elasticsearch с помощью RPM. Для установки Elasticsearch в различных средах необходимо выполнить указанную ниже команду, которая может отличаться для разных платформ:

- A. Для CentOS и более старых дистрибутивов на базе Red Hat:

```
sudo yum install elasticsearch
```

- B. Для Fedora и других более новых дистрибутивов Red Hat:

```
sudo dnf install elasticsearch
```

- C. Для дистрибутивов на базе OpenSUSE:

```
sudo zypper install elasticsearch
```

Установка пакета RPM вручную

Elasticsearch можно установить вручную, используя пакет RPM, для этого необходимо выполнить следующие шаги.

1. Загрузить RPM-файл Elasticsearch 8.7.1 с помощью команды

```
wget https://artifacts.elastic.co/downloads/elasticsearch/
elasticsearch-8.7.1-x86_64.rpm
```

2. После загрузки RPM сравнить SHA с опубликованной контрольной суммой. Для сравнения выполнить следующие команды:

```
wget https://artifacts.elastic.co/downloads/elasticsearch/
elasticsearch-8.7.1-x86_64.rpm.sha512
shasum -a 512 -c elasticsearch-8.7.1-x86_64.rpm.sha512
```

3. После завершения загрузки и проверки контрольной суммы извлечь архив, выполнив команду

```
sudo rpm --install elasticsearch-8.7.1-x86_64.rpm
```

УСТАНОВКА ELASTICSEARCH В ОС WINDOWS

Чтобы установить Elasticsearch в операционной системе Windows, следуйте приведенным ниже инструкциям.

1. Загрузите ZIP-пакет Elasticsearch с официального сайта Elasticsearch. Перейдите на страницу загрузки (<https://www.elastic.co/downloads/elasticsearch>) и выберите вариант Windows ZIP для нужной версии Elasticsearch, например 8.7.1.
2. После загрузки ZIP-пакета распакуйте его содержимое в выбранную директорию. Например, создайте папку с именем `elasticsearch` и распакуйте в нее ZIP-файл.
3. Перейдите в папку `"config"` в каталоге Elasticsearch. Откройте файл `elasticsearch.yml` в текстовом редакторе, чтобы изменить параметры конфигурации в соответствии с вашими требованиями. Вы можете настроить такие параметры, как имя кластера, имя узла, сетевые настройки и др.
4. Откройте командную строку с правами администратора. Перейдите в каталог Elasticsearch bin, выполнив следующую команду:

```
cd <путь_к_elasticsearch>\bin
```

5. Замените путь `<path_to_elasticsearch>` фактическим путем, по которому вы извлекли ZIP-пакет Elasticsearch.

6. Запустите Elasticsearch, выполнив следующую команду:

```
elasticsearch.bat
```

Это запустит сервер Elasticsearch на локальной машине.

ЗАПУСК И ПРОВЕРКА СЛУЖБЫ ELASTICSEARCH

Чтобы проверить, запущен ли узел Elasticsearch, можно отправить запрос HTTP GET на хост и порт Elasticsearch. Например, установку Elasticsearch на локальной машине можно проверить, перейдя по следующему URL:

```
curl -XGET "http://localhost:9200/"
```

Если Elasticsearch запущен, на указанный URL-адрес будет получен следующий ответ:

```
1. {
2.   "name" : "MacBook-Pro-10.local",
3.   "cluster_name" : "elasticsearch" ,
4.   "cluster_uuid" : "1qXgBTiAS528an81yRX2jg",
5.   "version" : {
6.     "number" : "8.7.1",
7.     "build_flavor" : "default",
8.     "build_type" : "tar",
9.     "build_hash" : "f229ed3f893a515d590d0f39b05f68913e2d9b53",
10.    "build_date" : "2023-04-27T04:33:42.127815583Z",
11.    "build_snapshot" : false,
12.    "lucene_version" : "9.5.0",
13.    "minimum_wire_compatibility_version" : "7.17.0",
14.    "minimum_index_compatibility_version" : "7.0.0"
15.  },
16.  "tagline" : "You Know, for Search"
17. }
```

ELASTICSEARCH REST API

Elasticsearch предоставляет обширный набор REST API, позволяющих взаимодействовать с системой по HTTP с помощью файлов JSON. Эти REST API служат для выполнения различных операций, таких как получение информации о состоянии индекса, кластера или шарда, изменение настроек, создание или удаление индекса, добавление или удаление документов. По сути, с их помощью можно выполнять любые операции.

REST API в Elasticsearch поддерживают ряд индексов. Указав в качестве параметра список имен индексов, разделенных запятыми, можно выполнять операции с этими конкретными индексами. Кроме того, в качестве значения

параметра можно использовать `_all`, чтобы указать, что запрос должен быть выполнен по всем индексам. Чтобы исключить конкретные индексы, можно поставить перед именем индекса знак «минус». Параметр `index` также поддерживает имена с подстановочными знаками, что позволяет сопоставлять несколько индексов на основе шаблона.

Чтобы улучшить читаемость ответа JSON, можно добавить `pretty=true` к любому запросу REST API. Это позволит представить вывод JSON в удобном для чтения формате. В качестве альтернативы, если нам нужен ответ в формате YAML, можно добавить в запрос параметр `format=yaml`. Теперь рассмотрим некоторые операции, которые можно выполнить с помощью Elasticsearch REST API.

CAT API

Elasticsearch предлагает CAT API, который позволяет получить конкретные данные в более удобном для чтения формате, чем необработанные данные JSON. Это особенно полезно, когда требуется просмотреть состояние кластера, индексов, шардов и другие важные показатели. CAT API позволяет извлечь данные о состоянии системы, список индексов и т. д. С помощью CAT API легко получить доступ к нужной информации в более удобной для пользователя форме.

Параметры CAT API

CAT API в Elasticsearch поддерживает набор общих параметров, расширяющих его функциональность. Параметр `verbose` позволяет включать в вывод заголовки для лучшей читаемости. Используя параметр `help`, можно получить подробную информацию о доступных полях для конкретной команды. С помощью параметра `headers` можно указать имена полей, которые будут включены в вывод. Параметр `response format` позволяет изменить формат вывода. Параметр `sort` позволяет упорядочить данные в определенном порядке. Эти параметры повышают удобство использования CAT API.

Verbose

При выполнении команд в Elasticsearch результаты по умолчанию выводятся без заголовков, что может затруднить их интерпретацию. Однако, используя опцию `verbose`, можно включить в результаты заголовки для более четкого и структурированного представления данных. Эта опция помогает идентифицировать поля и соответствующие им значения, повышая читаемость и понятность вывода команды. Например, если выполнить команду

```
curl -XGET "http://localhost:9200/_cat/nodes"
```

получим такой результат:

```
127.0.0.1 20 100 11 2.63 cdfhilmrstw * MacBook-Pro-10.local
```

В этом выводе определить все поля нелегко, поэтому добавим в параметр опцию `verbose`. Для этого к конечной точке API припишем `?v`, тогда запрос будет выглядеть так:

```
curl -XGET "http://localhost:9200/_cat/nodes?v"
```

После выполнения получим следующий результат:

```
1. ip heap.percent ram.percent cpu load_1m load_5m load_15m node.
   role master name
2. 127.0.0. 22 100 14 1.78 cd-.
   fhilmrstw * MacBook-Pro-10.local
```

В первоначальном ответе на запрос видим такие детали, как IP-адрес, процент кучи, процент оперативной памяти, загрузка процессора, роль узла, имя мастера и т. д. Заголовок не будет виден, если не использовать `?v`.

Help

Если передать параметр `help` в любой команде, она выведет подробную информацию о полях. Например, если выполнить команду

```
curl -XGET "http://localhost:9200/_cat/master?v"
```

получим такой результат:

```
1. id                  host      ip          node
2. rf_oBLg7Qzqk1CZ-72o95Q 127.0.0.1 127.0.0.1 MacBook-Pro-10.local
```

Теперь добавим `help` в качестве параметра, чтобы увидеть подробную информацию о поле. Запишем URL-адрес API следующим образом:

```
curl -XGET "http://localhost:9200/_cat/master?v&help"
```

После выполнения получим результат:

```
1. id |     | node id
2. host | h | host name
3. ip  |     | ip address
4. node | n | node name
```

Теперь мы видим описание полей заголовка. Это помогает понять, какой ответ возвращает команда.

Заголовки

Можно передать поля заголовка в каждой из команд CAT, чтобы показать в выводе только эти поля. Например, если из команды `nodes` нам нужен только IP-адрес и процессор, напишем следующую команду:

```
curl -XGET "http://localhost:9200/_cat/nodes?v&h=ip,cpu"
```

После ее выполнения получим такой результат:

```
1. ip          cpu
2. 127.0.0.1  13
```

В ответе видим нужные столбцы заголовков `ip` и `CPU`, для этого они были указаны в запросе. Таким образом, если нам нужны только определенные поля, их можно передать с помощью параметра `h`.

Форматы ответов

Можно изменить формат вывода API, указав это в соответствующем параметре. Доступны форматы JSON, text, YAML, smile или cbor в зависимости от требований. Например, список индексов можно вывести в текстовом формате с помощью следующей команды:

```
curl -XGET "http://localhost:9200/_cat/indices?v"
```

В результате ее выполнения получим такой результат:

```
1. health status index uuid pri
   rep docs.count docs.deleted store.size pri.store.size
2. green open .fleet-file-data-agent-000001 e4p2U5NwSA2IeETGnHBEqw
   1 0 0 225b 225b
3. green open .fleet-files-agent-000001 MzLplpbATXmFxRkwYe2WYg
   1 0 0 225b 225b
```

Теперь изменим формат на JSON, чтобы вывести тот же результат в JSON. Команда будет выглядеть так:

```
curl -XGET "http://localhost:9200/_cat/indices?format=json"
```

После ее выполнения получим следующий результат:

```
1. [
2.   {
3.     "health": "green",
4.     "status": "open",
```

56 Глава 2. Установка Elasticsearch

```
5.   "index": ".fleet-file-data-agent-000001",
6.   "uuid": "e4p2U5NwSA2IeETGnHBEqw",
7.   "pri": "1",
8.   "rep": "0",
9.   "docs.count": "0",
10.  "docs.deleted": "0",
11.  "store.size": "225b",
12.  "pri.store.size": "225b"
13. },
14. {
15.   "health": "green",
16.   "status": "open",
17.   "index": ".fleet-files-agent-000001",
18.   "uuid": "MzLplpbATXmFxRkwYe2WYg",
19.   "pri": "1",
20.   "rep": "0",
21.   "docs.count": "0",
22.   "docs.deleted": "0",
23.   "store.size": "225b",
24.   "pri.store.size": "225b"
25. }
26. ]
```

Этот же результат можно преобразовать в YAML, изменив параметр формата на YAML. Получим следующее:

```
1. "health": "green"
2. "status": "open"
3. index: ".fleet-file-data-agent-000001"
4. uuid: "e4p2U5NwSA2IeETGnHBEqw"
5. pri: "1"
6. rep: "0"
7. docs.count: "0"
8. docs.deleted: "0"
9. store.size: "225b"
10. pri.store.size: "225b"
11. - health: "green"
12. status: "open"
13. index: ".fleet-files-agent-000001"
14. uuid: "MzLplpbATXmFxRkwYe2WYg"
15. pri: "1"
16. rep: "0"
17. docs.count: "0"
18. docs.deleted: "0"
19. store.size: "225b"
20. pri.store.size: "225b"
```

Также поддерживаются форматы smile и cbor, которые представляют собой бинарную сериализацию общей модели данных JSON.

Сортировка

Каждая команда также поддерживает сортировку, для которой следует указать имя сортируемого поля и порядок сортировки. Для сортировки необходимо добавить ключевое слово `s` со знаком равенства, имя поля со знаком двоеточия и порядок сортировки. Команда будет выглядеть так:

```
Curl -XGET "http://localhost:9200/_cat/templates?v&s=version:-desc,order"
```

После ее выполнения получим следующий результат:

1. name	index_patterns	order	version
2. packetbeat-6.5.2	[packetbeat-6.5.2-*]	1	
3. packetbeat-6.2.2	[packetbeat-6.2.2-*]	1	
4. metricbeat-7.3.0	[metricbeat-7.3.0-*]	1	

Будет выведен список из предыдущего ответа, отсортированный с помощью ключевого слова `s`.

CAT COUNT API

Используя CAT COUNT API, можно подсчитать количество документов в одном индексе или во всех индексах. Формат CAT COUNT API выглядит так:

1. GET `/_cat/count/<index>`
2. GET `/_cat/count`

Первое выражение подсчитывает документы по одному индексу, а второе — по всем индексам. Например:

```
GET /_cat/count?v
```

Результат выполнения будет следующим:

1. epoch	timestamp	count
2. 1582645351	15:42:31	11824319

Таким образом можно получить общее количество документов по всем доступным индексам.

CAT HEALTH API

Информацию о состоянии кластера можно получить с помощью CAT HEALTH API. Он похож на API состояния кластера, а его формат выглядит так:

```
GET /_cat/health
```

Чтобы получить эту информацию, необходимо выполнить следующую команду:

```
curl -XGET ''http://localhost:9200/_cat/health?v&pretty"
```

Получим результат:

```
1. epoch timestamp cluster status node.total node.data  
shards pri relo init unassign pending_tasks max_task_wait_time ac-  
tive_shards_percent  
2. 1582646292 15:58:12 elasticsearch yellow 1 1  
200 200 0 0 87 0 -  
69.7%
```

В ответе видим общее количество узлов, узлы данных, информацию о шардах, ожидающие задания и т. д.

CAT INDICES API

Используя CAT INDICES API, можно получить высокоуровневую информацию об индексах кластера. Чтобы получить список всех индексов, необходимо выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cat/indices?v"
```

Получим результат:

```
1. health status index uuid  
pri rep docs.count docs.deleted store.size pri.store.size  
2. yellow open players ljylFwacRniKzW-  
2j1N-dPA 1 1 6 0 27.6kb 27.6kb  
3. yellow open mappingtest3 aQaGqoBARL-7dFY-  
Fgfak8A 1 1 2 0 8.4kb 8.4kb  
4. yellow open my_index_stan EYUj59KTSs6GznogUl-  
9dxA 1 1 0 0 283b 283b
```

Видим подробную информацию об индексах кластера, такую как состояние, статус, имя индекса, uuid, количество первичных шардов, количество реплик, количество документов, количество удаленных документов, размер хранилища и т. д.

CAT MASTER API

Используя этот API, можно получить информацию о главном узле. Для этого нужно выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cat/master?v"
```

Получим ответ:

```
1. id          host      ip        node
2. rf_oBLg7Qzqk1CZ-72o95Q 127.0.0.1 127.0.0.1 MacBook-Pro-10.local
```

Видим данные о главном узле кластера, такие как идентификатор узла, IP-адрес, имя кластера и т. д.

CAT NODES API

Используя этот API, можно получить подробную информацию об узлах кластера. Для этого нужно выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cat/nodes?v"
```

Получим ответ:

```
1. ip heap.percent ram.percent cpu load_1m load_5m load_15m
   node.role master name
2. 127.0.0.1 44 100 13 1.98 cd-
   fhlmrstw * MacBook-Pro-10.local
```

Видим подробную информацию об узле кластера, такую как IP-адрес, процент кучи, процент оперативной памяти, объем загрузки процессора и т. д.

CAT SHARDS API

API-интерфейс SHARD в CAT предоставляет подробную информацию об узлах и их шардах, например о том, являются ли они первичными или репликами, сколько байтов они заняли на диске и сколько документов включают. Чтобы получить информацию о шардах, нужно выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cat/shards?v"
```

Получим ответ:

```
1. index shard prirep state docs
   store ip node
2. chocolates 0 p STARTED 4
   14.6kb 127.0.0.1 ANURAGLPTP0305
3. chocolates 0 r UNASSIGNED
```

До сих пор мы рассматривали CAT API, предназначенные для получения информации в виде числовых значений в текстовом формате. Эту информацию получить легко. Теперь рассмотрим некоторые кластерные API.

API КЛАСТЕРОВ

API кластеров в Elasticsearch используются для выполнения операций и получения информации на уровне кластера. Эти API предоставляют ценные сведения об общем состоянии кластера, а также его отдельных узлов. Используя API на уровне кластера, можно собирать важные метрики, отслеживать производительность кластера и управлять параметрами его работы.

API состояния кластера

Мы можем получить информацию о состоянии кластера с помощью API состояния. Этот API возвращает результат в формате JSON, в отличие от CAT API. Чтобы получить информацию о состоянии кластера, нужно выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cluster/health"
```

Результат:

```
1. {
2.   "cluster_name" : "elasticsearch",
3.   "status" : "yellow",
4.   "timed_out" : false,
5.   "number_of_nodes" : 1,
6.   "number_of_data_nodes" : 1,
7.   "active_primary_shards" : 202,
8.   "active_shards" : 202,
9.   "relocating_shards" : 0,
10.  "initializing_shards" : 0,
11.  "unassigned_shards" : 87,
12.  "delayed_unassigned_shards" : 0,
13.  "number_of_pending_tasks" : 0,
14.  "number_of_in_flight_fetch" : 0,
15.  "task_max_waiting_in_queue_millis" : 0,
16.  "active_shards_percent_as_number" : 69.8961937716263
17. }
```

Таким образом, в ответе получаем JSON-документ, описывающий состояние кластера.

API статистики кластера

Этот API возвращает статистику кластера для получения различных данных, например метрики индексов и метрики узлов. В одном API мы получаем такие данные, как количество шардов, объем использования памяти, размер

хранилища, версии JVM, объем использования ЦП, версия ОС и т. д. Чтобы получить статистику кластера, нужно выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cluster/state/_all/twitter"
```

Получим такой документ JSON:

```
1. {
2.     "cluster_name" : "elasticsearch",
3.     "cluster_uuid" : "BIP_9t5fR-SxB72hLM8SwA",
4.     "version" : 37326,
5.     "state_uuid" : "6L8zreS5T7GFWPIHWu972Q",
6.     "master_node" : "OsuaWgbGQd2KXbWE3ENfEg",
7.     "blocks" : {
8.         "indices" : {
9.             "index" : {
10.                 "5" : {
11.                     "description" : " index read-only (api)",
12.                     "retryable" : false,
13.                     "levels" : [
14.                         "write",
15.                         "metadata_write"
16.                     ]
17.                 }
18.             }
19.         }
20.     },
21.     "nodes" : {
22.         "OsuaWgbGQd2KXbWE3ENfEg" : { {
23.             .....
24.         }
25.     },
26.     "metadata" : {
27.         "templates" : { },
28.         "indices" : {
29.             "twitter" : {
30.                 "state" : "open",
31.                 "settings" : {
32.                     " index" : {
33.                         .....
34.                     }
35.                 },
36.                 "mappings" : {
37.                     "_doc" : {
38.                         " properties" : {
39.                             .....
40.                         }
41.                     }
42.                 },
43.                 "aliases" : [ ],
44.             }
45.         }
46.     }
47. }
```

```
44.     "primary_terms" : {
45.         "0" : 125
46.     },
47.     "in_sync_allocations" : {
48.         "0" : [
49.             "F0PQ_WKwQbaFqDF1C9ix8w"
50.         ]
51.     }
52. }
53. },
54. "index-graveyard" : {
55.     "tombstones" : [ ]
56. }
57. },
58. "routing_table" : {
59.     "indices" : {
60.         "twitter" : {
61.             "shards" : {
62.                 "0" : [
63.                     .....
64.                 ]
65.             }
66.         }
67.     }
68. },
69. "routing_nodes" : {
70.     "unassigned" : [
71.         {
72.             .....
73.             "unassigned_info" : {
74.                 .....
75.             }
76.         }
77.     ],
78.     "nodes" : {
79.         "0suawgbGQd2KXbWE3ENfEg" : [
80.             {
81.                 .....
82.             }
83.         ]
84.     }
85. }
86. }
```

Результат содержит имя кластера, UUID, версию, сведения об узлах, метаданные и т. д.

Мы познакомились с несколькими основными API, которые можно использовать для получения нужной информации. Существует множество других API: для извлечения индекса, документов, поиска, снимков и т. д. Мы рассмотрим их в соответствующих главах книги.

ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели Elasticsearch версии 8, отметив его новые возможности и улучшения. Затем мы приступили к пошаговой установке Elasticsearch на разные платформы — Linux, macOS, Debian, RPM и Windows. Мы также рассмотрели, как проверить успешность установки, и обсудили REST API Elasticsearch.

В следующей главе мы сосредоточимся на Elastic Stack, комплексном наборе инструментов, включающем Elasticsearch, Logstash, Kibana и Beats. Мы представим обзор каждого компонента и рассмотрим, как они взаимодействуют друг с другом, расширяя возможности работы с данными.

ВОПРОСЫ

1. Что нового в Elasticsearch 8?
2. Как установить Elasticsearch на разные платформы?
3. Как перечислить все индексы кластера Elasticsearch?
4. Как получить количество всех документов в Elasticsearch?
5. Как получить информацию о состоянии кластера?
6. Как получить сведения об узлах кластера Elasticsearch?
7. Как получить статистику для кластера Elasticsearch?

ГЛАВА 3

Elastic Stack: экосистема Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы познакомились с новейшими возможностями Elasticsearch 8. Затем узнали, как установить Elasticsearch на разные платформы, и изучили Elasticsearch REST API. В этой главе мы подробно рассмотрим Elastic Stack и его основные компоненты: Elasticsearch, Logstash, Kibana и Beats. Сначала мы представим обзор каждого компонента, а затем подробнее разберем их функциональные возможности. Наконец, мы изучим, как эти компоненты работают вместе для решения различных задач.

СТРУКТУРА

В этой главе:

- Обзор компонентов Elastic Stack
- Elasticsearch: механизм поиска и аналитики
- Logstash: пайплайн обработки данных
- Kibana: инструмент визуализации данных
- Beats: легковесная доставка данных
- Интеграция компонентов Elastic Stack

ЦЕЛИ

По завершении этой главы вы получите полное представление об Elastic Stack, мощном наборе инструментов с открытым исходным кодом для хранения, поиска и анализа данных. Вы изучите компоненты, из которых состоит Elastic Stack, включая Elasticsearch для распределенного поиска и аналитики, Kibana для визуализации и изучения данных, Logstash для сбора и обработки данных и Beats для легковесной доставки данных. Кроме того, вы узнаете, как эти компоненты взаимодействуют друг с другом, позволяя эффективно управлять данными и анализировать их, извлекать полезную информацию и с легкостью создавать мощные приложения. Эти знания позволят использовать весь потенциал Elastic Stack и улучшить процессы принятия решений на основе данных.

ОБЗОР КОМПОНЕНТОВ ELASTIC STACK

Изначально известный как стек ELK, Elastic Stack включает в себя три приложения с открытым исходным кодом — Elasticsearch, Logstash и Kibana. Они работают вместе для решения разнообразных бизнес-задач. Название стека было изменено на Elastic Stack после появления новых инструментов, таких как Beats, APM и машинное обучение.

Elastic Stack представляет собой сквозное решение для анализа данных, включая их поиск, анализ и визуализацию. Он работает с разными типами данных, такими как структурированные, неструктурированные, основанные на времени, геоданные и любые другие. Beats или Logstash можно использовать для ввода данных в Elasticsearch, который является централизованным хранилищем и считается сердцем Elastic Stack.

После отправки данных в Elasticsearch их можно использовать для поиска и анализа. С помощью Kibana можно создавать визуализации и применять методы машинного обучения. Кроме того, Kibana позволяет настраивать APM для мониторинга производительности приложений или использовать технологию **управления информацией и событиями безопасности (SIEM)** для анализа безопасности.

На рис. 3.1 показано, как используется Elastic Stack, когда инструмент Beats настроен на сбор данных файлов и метрик с серверов.

Logstash настроен на сбор данных БД и отправку их в Elasticsearch, а Filebeat отправляет данные непосредственно в Elasticsearch. Можно настроить Beats на отправку данных непосредственно в Elasticsearch или использовать Logstash для обогащения или модификации данных перед их отправкой в Elasticsearch.

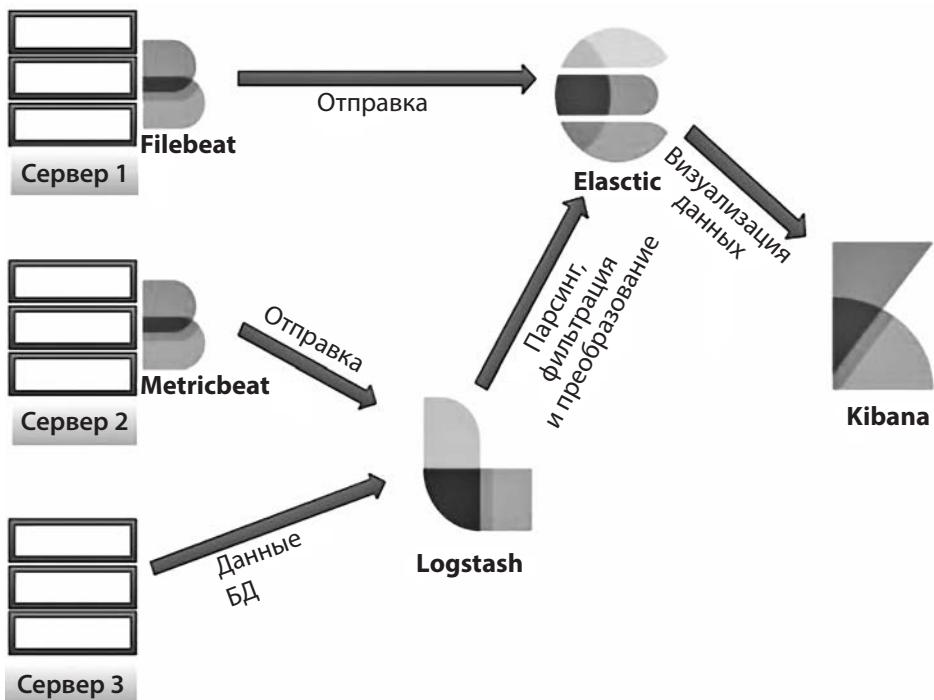


Рис. 3.1. Схема Elastic Stack

Теперь более подробно рассмотрим компоненты Elastic Stack.

ELASTICSEARCH: МЕХАНИЗМ ПОИСКА И АНАЛИТИКИ

Elasticsearch — основной компонент стека Elastic. Это мощный поисковый и аналитический движок, используемый для работы с разными типами данных. Он выстроен на базе поисковой библиотеки с открытым исходным кодом Apache Lucene, но отличается масштабируемостью и RESTful API для простоты использования. Elasticsearch может работать со структурированными и неструктурными данными, текстом, числовыми и геопространственными данными, то есть это универсальный инструмент для поиска и анализа больших объемов данных. Это распределенная система, функционирующая на нескольких узлах, что делает ее высокодоступной и отказоустойчивой. Elasticsearch — центральный компонент стека Elastic Stack, его обычно называют сердцем Elastic Stack. Он был впервые представлен компанией

Elasticsearch N.V. в 2010 году и с тех пор стал популярным инструментом поиска и аналитики.

Elasticsearch предназначен для работы с большими объемами данных и может использоваться для широкого спектра задач, включая поиск, ведение логов, обеспечение безопасности, бизнес-аналитику и многое другое. Распределенная природа позволяет ему масштабироваться горизонтально, с добавлением узлов в кластер по мере необходимости, чтобы обеспечить обработку растущего объема данных и нагрузки в виде запросов.

Для хранения и поиска данных Elasticsearch использует распределенное хранилище документов на основе JSON. Каждый документ индексируется и доступен для поиска, причем поиск может осуществляться по нескольким индексам и типам. Elasticsearch поддерживает широкий спектр поисковых запросов, от простого поиска по ключевым словам до сложной агрегации и анализа.

Помимо поиска, Elasticsearch предоставляет расширенные возможности аналитики, включая машинное обучение для обнаружения аномалий, прогнозирования и классификации. Он также интегрирован с такими популярными инструментами BI, как Tableau и Grafana.

Elasticsearch обладает высокой расширяемостью и обширной экосистемой плагинов и интеграций для разнообразных задач. Он подходит бизнесу любого размера, от небольших стартапов до крупных компаний, и является популярным инструментом для создания современных приложений, управляемых данными.

LOGSTASH: ПАЙПЛАЙН ОБРАБОТКИ ДАННЫХ

Logstash — важный компонент Elastic Stack, поскольку он предоставляет мощный и гибкий пайплайн обработки данных. Пайплайн получает данные из различных источников и выполняет их преобразование с помощью плагинов ввода, фильтрации и вывода. Плагины ввода помогают считывать данные из файлов, баз данных и систем обмена сообщениями, например Kafka. С помощью плагинов фильтрации можно изменять данные перед их отправкой получателю. Эти плагины позволяют выполнять такие задачи, как удаление или добавление полей, изменение типов данных и преобразование неструктурированных данных в структурированные. Плагины вывода записывают преобразованные данные в места назначения, такие как Elasticsearch, файлы, базы данных, системы обмена сообщениями, например Kafka, и т. д.

Одно из существенных преимуществ использования Logstash состоит в том, что он поддерживает широкий спектр плагинов ввода и вывода данных, что

упрощает интеграцию с разными системами. Кроме того, он позволяет обрабатывать данные в реальном времени и в масштабе, что делает его идеальным инструментом для обработки больших объемов данных. Logstash подходит для следующих целей:

- анализ структурированных и неструктурированных данных и событий;
- соединение с различными входными и выходными источниками, такими как файлы, Beats, Kafka, базы данных, Elasticsearch и т. д.;
- преобразование данных и их хранение для анализа;
- получение данных из различных источников, таких как базы данных, CSV и другие файлы.

Logstash также поставляется с готовыми фильтрами и кодеками, которые можно использовать для решения общих задач преобразования данных. Кроме того, можно писать собственные плагины, используя набор средств разработки, предоставляемый Logstash. Такая гибкость делает его универсальным инструментом, подходящим для всевозможных задач обработки данных. На рис. 3.2 показан пайпайн Logstash.

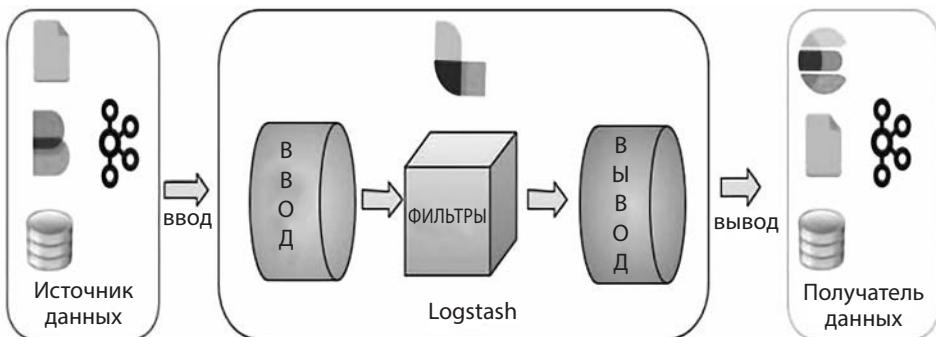


Рис. 3.2. Logstash

На рисунке представлены три блока, где первый блок — источник, из которого Logstash получает входные данные. Источником являются различные инструменты и программы, например Kafka, Beats, File, Elasticsearch, S3, базы данных и т. д., которые могут отправлять данные в Logstash. Далее следует пайплайн Logstash с плагинами ввода, фильтрации и вывода. Данные поступают через plugin ввода, который считывает их из различных источников. Затем plugin фильтрации преобразует данные и передает в plugin вывода, который отправляет их к получателю (третий блок). Получателем данных могут служить различные инструменты и программы, такие как Kafka, Beats, File, базы данных, Elasticsearch, S3, HTTP, NoSQL и т. д.

Плагины ввода данных Logstash

Плагины ввода Logstash используются для чтения данных. Они подключаются к различным источникам входных данных, таким как файлы, базы данных, очереди сообщений и другие системы хранения данных.

Logstash предоставляет широкий спектр плагинов для ввода данных, включая, помимо прочего:

- **Stdin** — считывает данные стандартного ввода.
- **Beats** — считывает данные из Beats, легковесного отправителя данных в наборе Elastic Stack.
- **CloudWatch** — считывает логи из Amazon CloudWatch.
- **GitHub** — считывает события из веб-хука GitHub.
- **RabbitMQ** — считывает сообщения из очереди RabbitMQ.
- **SQLite** — считывает данные из базы данных SQLite.
- **TCP** — считывает данные из сокета TCP.
- **Twitter** — считывает данные из Twitter Streaming API.
- **File** — считывает данные из файлов на диске.
- **MongoDB** — считывает данные из базы данных MongoDB.
- **Elasticsearch** — считывает данные из Elasticsearch.
- **Redis** — считывает данные из Redis.
- **Kafka** — считывает данные из Kafka.
- **Http** — считывает данные с конечной точки HTTP.
- **JDBC** — используется для чтения данных из баз данных, совместимых с JDBC, таких как MySQL, Oracle и PostgreSQL.
- **S3** — используется для чтения данных из сегментов Amazon S3.
- **Syslog** — используется для чтения данных с серверов Syslog, собирающих системные данные.

Это лишь некоторые наиболее важные из плагинов ввода Logstash. Плагины ввода позволяют Logstash собирать данные из различных источников и нормализовывать их в общий формат для дальнейшей обработки и анализа.

Плагины фильтрации Logstash

Используя плагины фильтрации Logstash, можно выполнять преобразование исходных данных и множество других операций, например преобразование

неструктурированных сложных данных в простой структурированный формат, по которому легко осуществлять поиск. Также можно добавлять данные, например геоданные, с помощью поля IP-адреса, или удалять ненужные поля данных. Logstash предоставляет широкий спектр фильтров для преобразования и модификации данных, проходящих через пайпайн. Вот некоторые часто используемые плагины:

- **Grok** — парсит неструктурированные данные логов и извлекает поля с помощью регулярных выражений.
- **Date** — парсит и форматирует метки времени в данных событий.
- **Mutate** — выполняет операции на уровне полей, такие как переименование, удаление, преобразование типов данных и др.
- **JSON** — парсит и манипулирует данными JSON в событии.
- **CSV** — парсит и преобразует данные в формате CSV.
- **GeoIP** — добавляет информацию о геолокации на основе IP-адресов.
- **Translate** — выполняет поиск и перевод на основе предопределенных словарей.
- **XML** — парсит и манипулирует данными XML в рамках события.
- **UserAgent** — извлекает информацию из строк пользовательского агента.
- **KV** — извлекает пары «ключ — значение» из строкового поля.
- **Drop** — отбрасывает события, соответствующие заданным условиям.
- **Клонировать** — дублирует события и добавляет их в пайпайн.
- **Grokdissect** — парсит строки лога на пары «ключ — значение» по заданному шаблону.

Эти плагины фильтров, а также другие плагины, доступные в Logstash, предоставляют богатые возможности для модификации и улучшения данных перед их отправкой получателю.

Плагины вывода данных Logstash

Logstash предоставляет набор плагинов вывода для отправки данных в различные места назначения, и их можно использовать в соответствии с заданными требованиями. Например, если требуется отправить данные в Elasticsearch, следует использовать плагин Elasticsearch, а для отправки данных в файл следует использовать плагин File. Вот некоторые часто используемые плагины вывода:

- **Elasticsearch** — отправляет данные в кластер Elasticsearch для индексации и хранения.

- **File** — записывает события в файлы локальной или удаленной файловой системы.
- **Kafka** — публикует события в Apache Kafka, распределенную потоковую платформу.
- **STDOUT** — отправляет события на стандартный вывод (консоль).
- **Redis** — отправляет события в Redis, хранилище структур данных в памяти.
- **Amazon S3** — загружает события в Amazon Simple Storage Service для долговременного хранения.
- **Amazon Kinesis** — плагин потоковой передачи событий в Amazon Kinesis, полностью управляемую службу потоковой передачи данных.
- **Amazon CloudWatch** — отправляет события в Amazon CloudWatch для мониторинга и анализа.
- **Graphite** — отправляет события на сервер Graphite для мониторинга и построения графиков.
- **JDBC** — записывает события в реляционную базу данных с помощью **Java Database Connectivity (JDBC)**.
- **Logz.io** — отправляет события на Logz.io, облачную платформу управления логами и аналитики.
- **Splunk** — передает события в Splunk, популярный инструмент управления и анализа логов.
- **InfluxDB** — записывает события в InfluxDB, базу данных временных рядов.

Это лишь несколько плагинов вывода, доступных в Logstash. Вы можете выбрать подходящий плагин в зависимости от своей цели и настроить его на отправку обработанных данных соответствующим образом.

Существует множество других плагинов, которые помогают в различных сценариях использования Logstash. Теперь, когда мы обсудили плагины ввода, фильтрации и вывода, посмотрим, как с их помощью создать пайплайн для импорта, фильтрации и отправки данных в место назначения. Для этого необходимо создать файл конфигурации Logstash в формате, показанном ниже:

```
1. input
2. {
3. ....
4. }
5. filter
6. {
7. ....
8. }
9. output
```

```
10. {  
11.     .....  
12. }
```

Представленный формат конфигурации Logstash содержит блоки ввода, фильтрации и вывода. В блоке ввода необходимо указать имя плагина ввода для чтения данных из нужного источника. Аналогично в блоке фильтрации указывается имя соответствующего плагина фильтрации для преобразования данных. Наконец, в блоке вывода указывается имя плагина вывода для передачи данных по назначению. Соблюдая эту структуру, можно создать файл конфигурации Logstash и использовать его для запуска процесса ввода данных. Протестировать пайпайн Logstash без создания файла конфигурации можно, используя следующую команду:

```
bin/logstash -e 'input { stdin { } } output { stdout { } }'
```

Тестируем пайпайн Logstash с помощью этой команды легко, поскольку таким образом плагин ввода предоставляетя в качестве стандартного ввода, а плагин вывода — в качестве стандартного вывода. Выполнив эту команду, можно написать любой текст, который Logstash получит через плагин `stdin`, и вывести его на экран с помощью плагина вывода `stdout`. На следующем снимке экрана показано выполнение этой команды:

Рис. 3.3. Ответ на запуск пайплайна Logstash

На рис. 3.3 представлен пайпайн Logstash, использующий `stdin` в качестве плагина ввода и `stdout` в качестве плагина вывода. Он выводит данные,

получаемые стандартным способом с клавиатуры. Здесь была выполнена следующая команда:

```
bin/logstash -e 'input { stdin { } } output { stdout {} }'
```

После успешного выполнения Logstash ожидает ввода данных пользователем. После ввода Logstash отображает результат и дополнительные сведения, такие как метка времени и имя хоста. Ниже приведен пример ответа:

```
1. The stdin plugin is now waiting for input:
2. [2023-07-22T16:20:35,864][INFO ][logstash.agent ] Pipe-
   lines running {:count=>1, :running_pipelines=>[:main], :non_run-
   ning_pipelines=>[]}
3. Sophia Julian
4. {
5.     "message" => " Sophia Julian",
6.     "host" => {
7.       "hostname" => "MacBook-Pro-10.local"
8.     },
9.     "event" => {
10.    "original" => " Sophia Julian"
11.  },
12.   "@timestamp" => 2023-07-22T10:50:58.611100Z,
13.   "@version" => "1"
14. }
```

Таким образом можно протестировать пайплайн Logstash, но для этого необходимо создать файл конфигурации Logstash и сохранить его в каталоге `/logstash/conf.d/` для обработки входных и выходных данных. Например, если файл конфигурации требуется для создания одного пайплайна ввода и вывода, можно создать файл и скопировать в него код блока ввода и вывода. Допустим, имя файла — `standard_input_output.conf`. Тогда код файла конфигурации Logstash будет следующим:

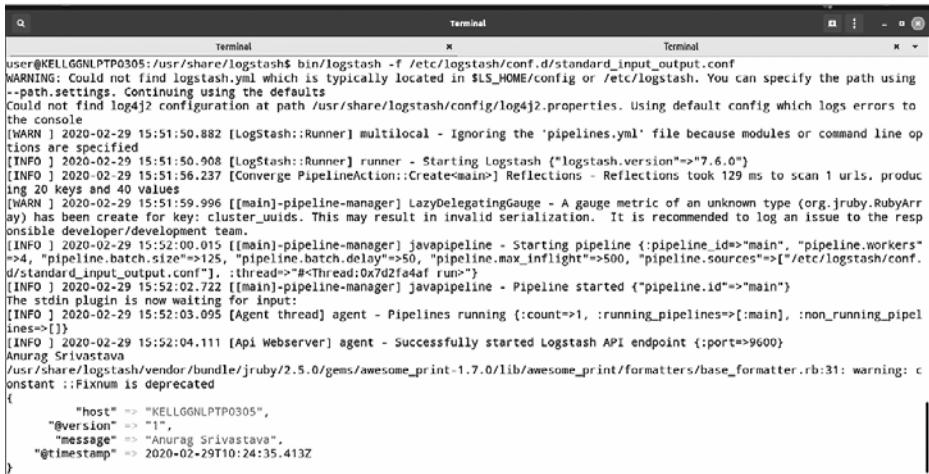
```
1. input
2. {
3.   stdin { }
4. }
5. output
6. {
7.   stdout {}
8. }
```

Запустить файл конфигурации Logstash можно, выполнив следующую команду:

```
bin/logstash -f /etc/logstash/conf.d/standard_input_output.conf
```

Приведенная выше команда выполняется в операционной системе Ubuntu, а каталог `conf.d` может отличаться в зависимости от операционной системы.

Итак, можно запустить пайпайн Logstash с помощью команды выше (рис. 3.4).



```

Terminal
user@KELLOGGNLPTP0305:/usr/share/logstash$ bin/logstash -f /etc/logstash/conf.d/standard_input_output.conf
WARNING: Could not find logstash.yml which is typically located in $LS_HOME/config or /etc/logstash. You can specify the path using --path.settings. Continuing using the defaults
Could not find log4j2 configuration at path /usr/share/logstash/config/log4j2.properties. Using default config which logs errors to the console
[WARN ] 2020-02-29 15:51:50.882 [Logstash::Runner] multilocal - Ignoring the 'pipelines.yml' file because modules or command line options are specified
[INFO ] 2020-02-29 15:51:50.908 [Logstash::Runner] runner - Starting Logstash ("logstash.version">"7.6.0")
[INFO ] 2020-02-29 15:51:56.237 [Converge PipelineAction::Create<main>] Reflections - Reflections took 129 ms to scan 1 urls, producing 20 keys and 40 values
[WARN ] 2020-02-29 15:51:59.996 [[main]-pipeline-manager] LazyDelegatingGauge - A gauge metric of an unknown type (org.jruby.RubyArr
ay) has been created for key: cluster_uids. This may result in invalid serialization. It is recommended to log an issue to the responsible developer/development team.
[INFO ] 2020-02-29 15:52:00.015 [[main]-pipeline-manager] javapipeline - Starting pipeline {:pipeline_id=>"main", "pipeline.workers"
=>4, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>50, "pipeline.max_inflight"=>500, "pipeline.sources"=>["/etc/logstash/conf.d/standard_input_output.conf"], :thread=>"#<Thread:0x7d2f4af run=?"}
[INFO ] 2020-02-29 15:52:02.722 [[main]-pipeline-manager] javapipeline - Pipeline started {"pipeline.id"=>"main"}
The stdin plugin is now waiting for input:
[INFO ] 2020-02-29 15:52:03.095 [Agent thread] agent - Pipelines running {:count=>1, :running_pipelines=>[:main], :non_running_pipelines=>[]}
[INFO ] 2020-02-29 15:52:04.111 [Api Webservice] agent - Successfully started Logstash API endpoint {:port=>9600}
Anurag Srivastava
/usr/share/logstash/vendor/bundle/jruby/2.5.0/gems/awesome_print-1.7.0/lib/awesome_print/formatters/base_formatter.rb:31: warning: c
onstant ::Fixnum is deprecated
{
    "host" => "KELLOGGNLPTP0305",
    "@version" => "1",
    "message" => "Anurag Srivastava",
    "@timestamp" => 2020-02-29T10:24:35.413Z
}

```

Рис. 3.4. Ответ на запуск файла конфигурации Logstash

На рис. 3.4 показан пайпайн Logstash, использующий файл конфигурации Logstash. Для обработки разных входных и выходных данных можно создавать разные файлы конфигурации. Они будут выполняться при запуске Logstash, или их можно выполнить вручную с помощью команды выше.

Давайте теперь изучим еще один важный компонент Elastic Stack — Kibana.

КИБАНА: ИНСТРУМЕНТ ВИЗУАЛИЗАЦИИ ДАННЫХ

Kibana — это мощный инструмент с открытым исходным кодом, который расширяет функциональность Elasticsearch, предоставляя удобный интерфейс для визуализации и анализа данных. С помощью набора функций Kibana можно получить представление о данных Elasticsearch. Одна из ключевых функций Kibana — Discover; она позволяет исследовать данные, хранящиеся в Elasticsearch: выбирать поля, применять фильтры и искать данные по разным критериям. Эта функция облегчает ориентирование в данных и их понимание.

Визуализация — еще одна важная область применения Kibana. Этот инструмент позволяет создавать визуально привлекательные и интерактивные графики, диаграммы и дашборды. Визуализации можно настраивать для представления данных в разных форматах, таких как гистограммы, круговые диаграммы, линейные графики и др. Комбинируя визуализации на дашборде, можно создавать комплексное представление данных. Kibana также предлагает дополнительные

функции с помощью плагинов. **Мониторинг производительности приложений (Application Performance Monitoring, APM)** позволяет отслеживать производительность приложений. С помощью Dev Tools можно напрямую выполнять запросы к Elasticsearch в интерфейсе Kibana. Мониторинг стека, Stack Monitoring, дает представление о производительности всего стека Elastic. Canvas предлагает креативный способ представления данных с помощью настраиваемых дизайнов и макетов.

Машинное обучение в Kibana позволяет использовать продвинутые алгоритмы для выявления аномалий в данных, обнаружения закономерностей и предсказания тенденций. Это может быть особенно полезно для сценариев прогнозной аналитики и обнаружения аномалий.

Kibana SIEM – это ценная функция, которая помогает анализировать данные, связанные с безопасностью. Она позволяет исследовать потенциальные угрозы, собирать доказательства и даже может интегрироваться с системами тикетов для реагирования на инциденты.

Мониторинг времени работы в Kibana позволяет следить за доступностью и производительностью сервисов, обеспечивая их ожидаемую работоспособность.

В целом Kibana предлагает полный набор функций, позволяющих анализировать и визуализировать данные, хранящиеся в Elasticsearch, что делает ее незаменимым инструментом для исследования и анализа данных в экосистеме Elastic Stack. После установки получить доступ к Kibana можно через порт 5601, поэтому для открытия Kibana необходимо открыть следующий URL:

`http://localhost:5601`

При первых настройках безопасности мы увидим экран входа в Kibana (рис. 3.5).

После входа в систему Kibana предлагает ряд опций в следующих категориях:

- **Observability** (наблюдаемость). В этой категории представлены инструменты для наблюдения и мониторинга приложений, логов и метрик: **APM** для мониторинга производительности приложений, **Logs** для анализа логов и **Metrics** для мониторинга системных метрик.
- **Security** (безопасность). Функции безопасности Kibana позволяют реализовать надежную систему безопасности. Категория **Security** включает опцию **SIEM**, которая позволяет обнаруживать и анализировать угрозы для выявления потенциальных проблем безопасности и реагирования на них.
- **Visualize and Explore Data** (визуализация и исследование данных). Эта категория включает в себя широкий спектр инструментов. В нее входят **APM**, позволяющий получить подробную информацию о производительности приложений; **Canvas** – инструмент для креативного представления данных; **Dashboard** – инструмент для создания пользовательских дашбордов;

Discover — инструмент для изучения и поиска данных; **Graph** — инструмент для визуализации взаимосвязей между точками данных; **Logs** — инструмент для анализа данных логов; **Machine Learning** — инструмент для автоматического обнаружения аномалий и прогнозирования тенденций; **Maps** — инструмент для визуализации геопространственных данных; **Metrics** — инструмент для мониторинга показателей уровня системы; **SIEM** — инструмент для анализа безопасности; **Uptime** — инструмент для мониторинга доступности сервисов; **Visualize** — инструмент для создания различных типов визуализации.

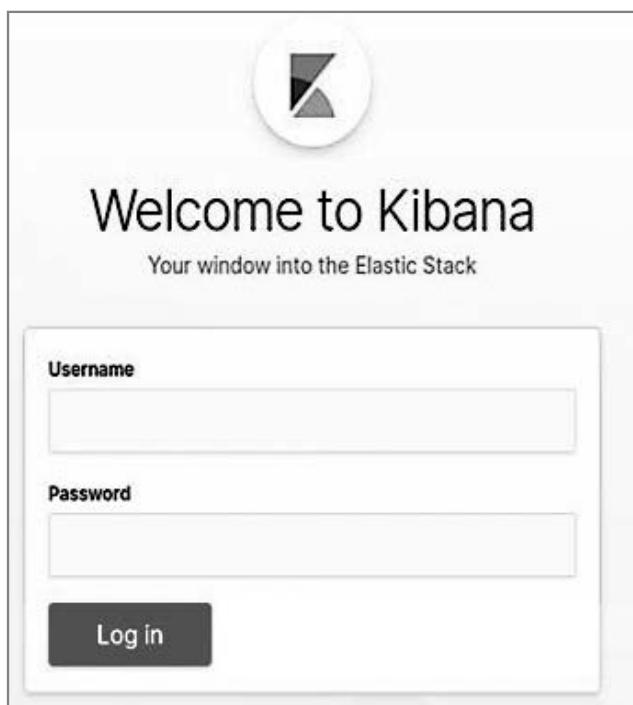


Рис. 3.5. Экран входа в систему Kibana

- Manage and Administer the Elastic Stack (управление и администрирование стека Elastic Stack). В эту категорию входят инструменты для управления и администрирования стека Elastic Stack. Сюда относятся **Console**, мощный интерфейс для выполнения запросов Elasticsearch; **Index Patterns** для управления шаблонами индексов, используемых для визуализации и анализа данных; **Monitoring** для мониторинга производительности Elastic Stack; **Rollups** для создания и управления заданиями сворачивания для обобщения данных; **Saved Objects** для управления и организации сохраненных визуализаций, дашбордов и поиска; **Security Settings** для настройки параметров безопасности;

Spaces для организации и разделения объектов Kibana; **Watchers** для создания и управления наблюдениями для целей оповещения и уведомления.

Эти опции в Kibana обеспечивают полный набор инструментов для анализа, визуализации, мониторинга и администрирования данных в Elastic Stack. На рис. 3.6 показан главный экран Kibana.

The screenshot displays the main dashboard of the Kibana application. At the top, there are four main sections: **Observability** (with APM, Logs, Metrics, and SIEM), **Upload data from log file**, **Use Elasticsearch data**, and **Visualize and Explore Data**. The **Visualize and Explore Data** section contains several cards: APM, Canvas, Dashboard, Discover, Graph, Logs, Machine Learning, Maps, Metrics, SIEM, Uptime, and Visualize. Below these cards are two large columns: **Manage and Administer the Elastic Stack** (Console, Index Patterns, Monitoring, Rollups, Saved Objects, Security Settings) and **Spaces** (Organizing dashboards and objects). At the bottom, there is a search bar and a link to view the full directory of Kibana plugins.

Рис. 3.6. Главная страница Kibana

На главной странице представлены важные инструменты и категории, к которым они относятся. Можно выбрать инструмент в зависимости от требований и сценария использования.

BEATS: ЛЕГКОВЕСНАЯ ОТПРАВКА ДАННЫХ

Beats – это одноцелевые инструменты для легковесной отправки данных в стеке Elastic Stack, которые собирают и отправляют данные разных типов из разных источников в Elasticsearch или Logstash для дальнейшей обработки и анализа. Beats разработаны с учетом простоты развертывания, эффективности и масштабируемости.

Существуют разные типы Beats, каждый из которых предназначен для конкретных сценариев использования:

- **Filebeat** – собирает и отправляет файлы логов и данные из файлов в Elasticsearch или Logstash.
- **Metricbeat** – собирает и отправляет метрики системного уровня и уровня приложения с серверов, служб и контейнеров.
- **Packetbeat** – захватывает и анализирует сетевой трафик в реальном времени, предоставляя информацию о сетевых протоколах, задержках и ошибках.
- **Winlogbeat** – собирает и отправляет логи событий Windows, облегчая мониторинг и анализ событий и логов, специфичных для Windows.
- **Auditbeat** – собирает данные аудита и следит за активностью пользователей и процессов в системах Linux.
- **Heartbeat** – контролирует доступность и время отклика служб и серверов, отправляя периодические запросы.

Beats обеспечивают простой и эффективный способ сбора и отправки данных, позволяя в реальном времени осуществлять мониторинг, анализ и визуализацию данных разных типов в рамках стека Elastic Stack. Из рис. 3.7 можно понять, как работают Beats в составе стека Elastic Stack.

Здесь показаны три инструмента Beats: **Filebeat**, **Metricbeat** и **Packetbeat**. Они отвечают за передачу данных файлов логов, системных метрик и данных сетевых пакетов в Elasticsearch. Затем Kibana использует эти данные для визуализации. Процесс установки Beats не рассматривается в этой книге, но можно обратиться к документации Elastic за подробными инструкциями.

Теперь рассмотрим каждый из этих инструментов более подробно.

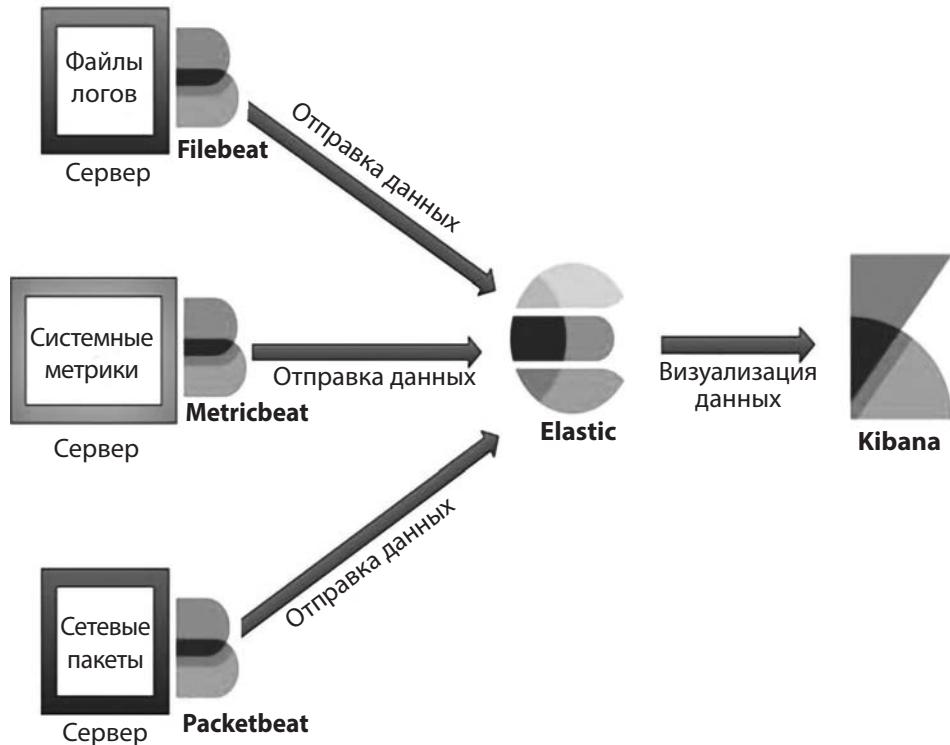


Рис. 3.7. Elastic Beats

Filebeat

Filebeat – это инструмент легковесной отправки данных, специально разработанный для чтения данных из файлов разного типа. Он эффективен при передаче логов и других данных. Как показано на рисунке выше, Filebeat отвечает за чтение данных из файлов логов и их транспортировку в кластер Elasticsearch. Настройка Filebeat, можно легко указать путь к директории файлов, которые требуется прочитать. Такая гибкость позволяет эффективно собирать и передавать данные из любых нужных файлов.

Настройка входящих данных

Настроить получение логов в Filebeat можно в файле `filebeat.yml` следующим образом:

1. `filebeat.inputs:`
2. `- type: log`

3. `paths:`
4. – `/var/log/messages`
5. – `/var/log/*.log`

В коде представлен блок входных данных Filebeat с типом лог. В разделе `paths` определяем сообщения и все файлы лога в каталоге `/var/log/`, используя подстановочный знак `*`, чтобы Filebeat читал все файлы лога в указанном каталоге. Кроме логов, можно указать в качестве входных данных Stdin, Kafka, UDP, TCP, S3, Syslog и т. д.

Filebeat поддерживает готовые плагины ввода для сбора данных логов из разных источников. Вот некоторые часто используемые:

- **Лог-файлы.** Плагин ввода логов используется для сбора данных логов из обычных текстовых лог-файлов. Это один из самых распространенных вариантов использования Filebeat.
- **Syslog.** Filebeat может собирать данные логов с серверов syslog. Его можно настроить на прослушивание сетевого порта или чтение лог-файлов, генерируемых syslog.
- **Логи событий Windows.** Плагин ввода `event_logs` используется для сбора данных логов событий Windows. Он позволяет указать, какие логи событий и идентификаторы событий следует собирать.
- **Контейнеры Docker.** Filebeat предоставляет плагин для сбора данных логов из контейнеров Docker. Он может отслеживать логи всех запущенных контейнеров.
- **Логи Kubernetes.** Плагин используется для сбора логов из подсистем и сервисов Kubernetes. Его можно настроить на автоматическое обнаружение подсистем и сбор логов.
- **Логи доступа и ошибок Apache.** Filebeat включает плагины ввода `apache2` для сбора логов доступа и ошибок Apache.
- **Логи доступа и ошибок Nginx.** Подобно Apache, Filebeat предлагает плагины для сбора логов доступа и ошибок Nginx.
- **Логи Redis.** Плагин может собирать логи экземпляров Redis. Он может быть настроен на чтение логов с серверов Redis или экземпляров Redis Sentinel.
- **Логи MySQL и PostgreSQL.** Filebeat включает плагины для сбора логов баз данных MySQL и PostgreSQL.
- **Логи AWS CloudWatch.** Плагин позволяет Filebeat собирать логи AWS CloudWatch.
- **Google Cloud Platform Pub/Sub.** Filebeat может читать логи GCP Pub/Sub с помощью плагина входящих данных `gcp_pubsub`.

Можно настроить Filebeat на использование соответствующего плагина в зависимости от источника данных лога.

Настройка исходящих данных

Filebeat можно настроить на вывод в определенный инструмент, прописав настройки исходящих данных в файле `filebeat.yml`. Filebeat поддерживает вывод в инструменты, описанные ниже.

Elasticsearch

Для вывода в Elasticsearch необходимо настроить блок `output.elasticsearch` с именем хоста. Если требуется переписать имя индекса Filebeat по умолчанию, можно задать пользовательское имя. См. следующий код:

```
1. output.elasticsearch:  
2.   hosts: ["https://localhost:9200"]  
3.   index: "filebeat-%{[agent.version]}-%{+yyyy.MM.dd}"
```

Если Elasticsearch защищен, нужно указать имя пользователя и пароль.

Logstash

Если требуется обработка данных перед индексацией в Elasticsearch, можно отправить данные Filebeat в Logstash. Используя вывод в Logstash, можно отправлять данные Filebeat в Logstash по протоколу lumberjack, который работает по TCP. Необходимо настроить блок `output.logstash`, указав данные о хосте, как в следующем коде:

```
1. output.logstash:  
2.   hosts: ["127.0.0.1:5044"]
```

Если существует более одного хоста Logstash, можно сбалансировать нагрузку на вывод, как показано ниже:

```
1. output.logstash:  
2.   hosts: ["localhost:5044", "localhost:5045"]  
3.   loadbalance: true  
4.   index: filebeat
```

Elastic Cloud

Для отправки событий в Elastic Cloud необходимо задать настройки в соответствующем блоке:

```
1. cloud.id: "staging:dXMtZWFzdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2M-".  
WE3NGJnMjRjZTMzYmI4ODExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI00WFmMGNjN-  
2Q3YTl10TYyNTc0Mw=="  
2. cloud.auth: "elastic:your_password".
```

В файл конфигурации Filebeat необходимо включить следующие параметры из приведенного выше кода:

- `cloud.id: "staging:dXMtZWfdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2MWE3NGJnMjRjZTMzYmI40DExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI0OWFmMGNjN2Q3YT1-10TYyNTc0Mw==".` Указывается идентификатор облака развертывания Elastic Cloud. Замените предоставленное значение фактическим идентификатором облака для развертывания.
- `cloud.auth: "elastic:your_password".` Указывает учетные данные для подключения к развертыванию Elastic Cloud. Замените "your_password" на действительный пароль пользователя elastic.

После настройки этих параметров Filebeat будет отправлять события в указанное развертывание Elastic Cloud. Убедитесь, что вы заменили заполнители на действительный идентификатор и пароль облака.

Redis

Используя опцию Redis output, можно отправлять события в Redis. Для этого необходимо настроить вот такую конфигурацию:

```
1. output.redis:
2.   hosts: ["localhost"].
3.   password: "your_password"
4.   key: "filebeat"
5.   db: 0
6.   timeout: 5
```

Следует настроить следующие параметры из кода выше:

- `output.redis`. Указывает плагин вывода Redis.
- `hosts: ["localhost"]`. Указывает хост(ы) Redis, к которому(ым) нужно подключиться. В этом примере мы подключаемся к экземпляру Redis, запущенному на локальной машине.
- `password: "your_password"`. Если экземпляр Redis требует аутентификации, можно указать здесь пароль. Замените "your_password" на действительный пароль.
- `key: "filebeat"`. Здесь указывается ключ Redis, под которым будут храниться события.
- `db: 0`. Указывает номер используемой базы данных Redis. В этом примере мы используем базу данных 0.
- `timeout: 5`. Устанавливает значение тайм-аута (в секундах) для соединения с Redis. В этом примере тайм-аут установлен на 5 секунд.

Настроив эти параметры в файле конфигурации Filebeat, можно успешно отправлять события в Redis.

Файл

Используя файловый вывод, можно отправлять события в файл формата JSON. Для вывода в файл нужно настроить блок `output.file`, как показано ниже:

```
1. output.file:  
2.   path: "/tmp/filebeat"  
3.   filename: filebeat
```

Kafka

Можно настроить блок вывода Kafka для отправки данных Filebeat в Apache Kafka. Для этого используйте следующий код в блоке `output.kafka`:

```
1. output.kafka:  
2.   # начальные брокеры для чтения метаданных кластера  
3.   hosts: ["host1_kafka:9092", "host2_kafka:9092", "host3_kafka:9092"]  
4.   # выбор темы сообщения + разбиение на разделы  
5.   topic: '%{[fields.topic_name]}'  
6.   partition.round_robin:  
7.   reachable_only: false  
8.   required_acks: 1  
9.     compression: gzip  
10.    max_message_bytes: 180000
```

С помощью приведенного кода настраиваются хосты сервера Kafka и топик для отправки сообщения.

Консоль

Используя вывод на консоль, можно записывать данные стандартным образом в формате JSON. Чтобы настроить блок `output.console`, в файл `filebeat.yml` надо добавить следующий код:

```
1. output.console:  
2.   pretty: true
```

Таким образом, выводить результаты Filebeat можно в Elasticsearch, Elastic Cloud, Logstash, Redis, файл, Kafka или консоль.

Metricbeat

Данные метрик можно получать из системы с помощью Metricbeat, легковесного инструмента отправки данных метрик. На рис. 3.7 показано, что Metricbeat настраивается на сервере, где отправляет метрики системы в централизованный

клuster Elasticsearch. Metricbeat предоставляет важные сведения, такие как данные об использовании процессора, памяти и доступном дисковом пространстве.

Настройка Metricbeat

Metricbeat можно настроить с помощью файла конфигурации `metricbeat.yml`, добавив опции в зависимости от поставленных целей.

Активация необходимых модулей

Для получения метрик от различных сервисов Metricbeat использует модули, и при необходимости собирать метрики определенного сервиса требуется активировать его модуль. Это можно сделать с помощью файла конфигурации или команды. Например, если требуется активировать модуль MySQL в Metricbeat, можно выполнить следующую команду для машин на базе Debian или RPM:

```
metricbeat modules enable mysql
```

Используя эту команду, можно активировать модуль MySQL и другие модули, такие как Apache, Nginx и т. д.

Конфигурация вывода

Данные Metricbeat можно отправлять в Elasticsearch или Logstash или настроить дашборд Kibana непосредственно с помощью Metricbeat. Чтобы отправлять метрики в Elasticsearch, необходимо добавить следующий код в файл конфигурации:

1. `output.elasticsearch:`
2. `hosts: ["localhost:9200"]`.

Чтобы отправлять данные в облако Elastic Cloud, следует добавить код, указывающий идентификатор Cloud ID:

1. `cloud.id: "staging:dxMtZWfdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2M-".WE3NGJnMjRjZTMzYmI4ODExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI0OWFmMGNjn-2Q3YT11OTyNTc0Mw=="`
2. `cloud.auth: "elastic:your_password"`.

Чтобы отправлять данные метрик в Logstash, нужно добавить такой код:

1. `output.logstash:`
2. `hosts: ["127.0.0.1:5044"]`

Чтобы настроить дашборд Metricbeat в Kibana, необходимо указать конечную точку Kibana в файле конфигурации Metricbeat. Это можно сделать с помощью следующего кода:

```
1. setup.kibana:  
2.   hosts: "localhost:5601"
```

Вместо localhost можно указать IP или URL установки Kibana. Таким образом, можно настроить Metricbeat на отправку данных метрик с разных машин и вывод результатов Metricbeat в Elasticsearch, Elastic Cloud или Logstash.

Packetbeat

Packetbeat — это легковесный инструмент, с помощью которого можно собирать данные сети. На рис. 3.7 выше показано, что Packetbeat настраивается на сервере, откуда отправляет данные о сетевых пакетах в кластер Elasticsearch. Kibana использует данные Beats для создания дашбордов и т. д.

Настройка Packetbeat

Packetbeat можно настроить, изменив файл конфигурации `packetbeat.yml`. Первым делом нужно сконфигурировать сетевой интерфейс для перехватывания трафика. Для этого требуется настроить параметр `packetbeat.interfaces.device`. Чтобы перехватывать трафик со всех источников, используйте следующее:

```
packetbeat.interfaces.device: any
```

После интерфейса требуется настроить протоколы для мониторинга. Стандартные протоколы уже указаны в файле конфигурации, и чтобы отслеживать нестандартные порты, нужно указать их в этом файле явно. Ниже приведен фрагмент файла конфигурации с указанием некоторых протоколов и их портов:

```
1. ===== Протоколы транзакций =====  
2. packetbeat.protocols:  
3.   - type: icmp  
4.     # Включить мониторинг ICMPv4 и ICMPv6. По умолчанию: false  
5.     enabled: true  
6.   - type: amqp  
7.     # Настроить порты, на которых будет прослушиваться трафик AMQP. Можно  
8.     # отключить протокол AMQP, закомментировав список портов.  
9.     ports: [5672]  
10.  - type: cassandra  
11.    # Port Cassandra для мониторинга трафика.  
12.    ports: [9042]  
13.  - type: dhcpcv4  
14.    # Настроить DHCP для портов IPv4.  
15.    ports: [67, 68]  
16.  - type: dns  
17.    # Настроить порты, на которых будет прослушиваться DNS-трафик. Можно  
18.    # отключить протокол DNS, закомментировав список портов.  
19.    ports: [53]
```

```
20. - type: http
21.   # Настроить порты, которые будут прослушивать HTTP-трафик. Можно
22.   # отключить протокол HTTP, закомментировав список портов.
23.   ports: [80, 8080, 8000, 5000, 8002]
```

В коде выше показан протокол и порты для него.

Данные Packetbeat можно отправлять в Logstash или непосредственно в Elasticsearch. Чтобы отправлять данные в Elastic Cloud, необходимо настроить идентификатор Cloud ID. См. код ниже:

```
1. cloud.id: "staging:dxMtZWFzdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2M-".
   WE3NGJmMjRjZTMzYmI4ODExYjg0Mjk0ZjRjNmMy2E2ZDA0MjI00WFmMGNjN-
   2Q3YT1lOTyNTc0Mw=="
2. cloud.auth: "elastic:your_password"
```

Чтобы отправлять данные в Elasticsearch, установленный на машине или в облаке, в файл конфигурации необходимо добавить следующий код:

```
1. output.elasticsearch:
2.   hosts: ["localhost:9200"]
```

Чтобы отправлять данные в Logstash, в файл конфигурации надо добавить такой код:

```
1. output.logstash:
2.   hosts: ["127.0.0.1:5044"]
```

Чтобы настроить дашборд Packetbeat, в файле конфигурации необходимо прописать конечную точку Kibana, как показано здесь:

```
1. setup.kibana:
2.   hosts: "localhost:5601"
```

Таким образом, данные Packetbeat можно выводить в Elasticsearch, Elastic Cloud или Logstash.

Winlogbeat

С помощью этого инструмента можно перехватывать логи событий Windows и следить за состоянием инфраструктуры на базе Windows. Winlogbeat помогает передавать логи событий Windows в Logstash и Elasticsearch.

Настройка Winlogbeat

Настроить Winlogbeat можно, изменив файл конфигурации `winlogbeat.yml`. В разделе событий файла конфигурации необходимо указать события, которые требуется отслеживать, как в следующем коде:

```

1. winlogbeat.event_logs:
2.   - name: Application
3.   - name: System
4.   - name: Security

```

Данные Winlogbeat можно отправлять в Logstash или непосредственно в Elasticsearch. Если требуется отправлять данные в Elastic Cloud, необходимо настроить Cloud ID, как показано ниже:

```

1. cloud.id:      "staging:dXMtZWfdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2M-".
WE3NGJnMjRjZTMzYmI40DExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI0OWFmMGNjN-
2Q3YTl10TYyNTc0Mw=="
2. cloud.auth: "elastic:your_password".

```

Чтобы отправлять данные в Elasticsearch, установленный на машине или в облаке, в файл конфигурации необходимо добавить такой код:

```

1. output.elasticsearch:
2.   hosts: ["localhost:9200"].

```

Чтобы отправлять данные в Logstash, в файл конфигурации необходимо добавить следующий код:

```

1. output.logstash:
2.   hosts: ["127.0.0.1:5044"]

```

Чтобы настроить дашборд Winlogbeat, в файле конфигурации необходимо прописать конечную точку Kibana, как показано здесь:

```

1. setup.kibana:
2.   hosts: "localhost:5601"

```

Таким образом, данные Winlogbeat можно выводить в Elasticsearch, Elastic Cloud или Logstash.

Auditbeat

Auditbeat — это легковесный инструмент, который собирает данные аудита с Linux-машин. Auditbeat позволяет отслеживать системы на базе Linux — процессы, действия пользователей и т. д. Auditbeat работает как auditd, взаимодействуя с фреймворком аудита Linux, собирая данные и отправляя события непосредственно в Elasticsearch или через Logstash.

Настройка Auditbeat

Сначала необходимо активировать модули, из которых требуется получить данные аудита. В следующем блоке кода показан системный модуль:

```

1. - module: system
2. datasets:
3.   - host    # Общая информация о хосте, например время работы, IP-адреса
4.   - login   # Вход пользователей в систему, выход из нее и загрузка
           # системы.
5.   - package # Установленные, обновленные и удаленные пакеты
6.   - process # Запущенные и остановленные процессы
7.   - socket  # Открытые и закрытые сокеты
8.   - user    # Информация о пользователе

```

Это конфигурация модуля по умолчанию; добавить или удалить нужный параметр можно в файле конфигурации.

Данные Auditbeat можно отправлять в Logstash или непосредственно в Elasticsearch. Чтобы отправлять данные в Elastic Cloud, необходимо настроить идентификатор Cloud ID. См. код ниже:

```

1. cloud.id: "staging:dXMtZWFzdC0xLmF3er5mb3VuZC5reyRjZWm2ZjI2M-".
   WE3NGJnMjRjZTMzYmI4ODExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI0OWFmMGNjN-
   2Q3YT11OTyNTc0Mw=="
2. cloud.auth: "elastic:your_password"

```

Чтобы отправлять данные в Elasticsearch, установленную на машине или в облаке, в файл конфигурации следует добавить такой код:

```

1. output.elasticsearch:
2.   hosts: ["localhost:9200"].

```

Чтобы отправлять данные в Logstash, в файл конфигурации надо добавить следующий код:

```

1. output.logstash:
2.   hosts: ["127.0.0.1:5044"]

```

Чтобы настроить дашборд Auditbeat, в файле конфигурации необходимо прописать конечную точку Kibana, как показано здесь:

```

1. setup.kibana:
2.   hosts: "localhost:5601"

```

Таким образом, данные Auditbeat можно выводить в Elasticsearch, Elastic Cloud или Logstash.

Heartbeat

Heartbeat используется для мониторинга работы службы; он может отслеживать статус их работы. Эта информация в Elastic Heartbeat отображается на специальном экране или выводится на настроенном для этой цели дашборде Kibana.

Инструмент Heartbeat может отслеживать любую веб-службу. Он использует протоколы ICMP, TCP и HTTP, а также поддерживает TLS.

Настройка Heartbeat

Настроить Heartbeat можно, изменив файл конфигурации `heartbeat.yml`. Например, для мониторинга службы HTTP с помощью Heartbeat можно выполнить такие настройки:

```
1. # /path/to/my.monitors.d/Localhost_service_check.yml
2. - type: http
3.   schedule: '@every 30s'
4.   hosts: ["http://localhost:80/servicename/statusendpoint"]
5.   check.response.status: 200
```

Используя приведенный выше код, можно проверять службу, например, каждые 30 секунд (настройки времени можно изменить в расписании) и проверять статус 200, определенный для поля `check.response.status`.

Аналогичным образом можно отслеживать TCP-порт:

```
1 - type: tcp
2. schedule: '@every 10s'
3. hosts: ["hostname"]
4. ports: [80, 9200, 5044]
```

Код выше мониторит порты **80, 9200 и 5044**.

Вывод Heartbeat можно отправить в следующие местоположения:

- Elasticsearch;
- Logstash;
- Elastic Cloud;
- Redis;
- файл;
- Kafka;
- консоль.

Чтобы отправлять данные в Elasticsearch, установленный на машине или в облаке, в файл конфигурации необходимо добавить следующий код:

```
1. output.elasticsearch:
2.   hosts: ["localhost:9200"].
```

Чтобы отправлять данные в облако Elastic Cloud, следует добавить код, указывающий идентификатор Cloud ID:

```

1. cloud.id: "staging:dXMTZWfdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2M-".
WE3NGJnMjRjZTMzYmI4ODExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI0OWFmMGNjN-
2Q3YT1lOTyNTc0Mw=="
2. cloud.auth: "elastic:your_password".

```

Чтобы отправлять данные в Logstash, в файл конфигурации надо добавить такой код:

```

1. output.logstash:
2.   hosts: ["127.0.0.1:5044"]

```

Таким образом, данные Heartbeat можно выводить в Elasticsearch, Elastic Cloud или Logstash. Мы не рассматриваем здесь другие варианты вывода, но вы можете обратиться к разделу конфигурации Filebeat за дополнительной информацией.

Functionbeat

Functionbeat — это инструмент Beat для бессерверной архитектуры, позволяющий получать данные с помощью Beat. Functionbeat может быть развернута как функция на платформе **Function-as-a-Service (FaaS)** облачного провайдера. После развертывания она начнет собирать и отправлять данные в Elasticsearch. После отправки в Elasticsearch данные можно будет проанализировать.

Настройка Functionbeat

Необходимо указать облачные сервисы, которые требуется отслеживать с помощью Functionbeat. Например, чтобы перехватывать события CloudWatch из AWS, необходимо задать такую конфигурацию:

```

1. functionbeat.provider.aws.endpoint: "s3.amazonaws.com"
2. functionbeat.provider.aws.deploy_bucket: "unique_bucket_name"
3. functionbeat.provider.aws.functions:
4.   - name: name_of_function
5.     enabled: true
6.     type: cloudwatch_logs
7.     description: "lambda function to capture the cloudwatch logs"
8.     triggers:
9.       - log_group_name: /aws/lambda/my-lambda-function

```

Таким образом, можно развернуть функцию на S3 для сбора логов CloudWatch.

Functionbeat можно развернуть в облаке любого провайдера, например AWS, Google и др. Так, чтобы развернуть его на AWS, используя среду Linux или Mac, необходимо написать следующий код:

```

1. export AWS_ACCESS_KEY_ID=aws_access_key_id
2. export AWS_SECRET_ACCESS_KEY=aws_secret_access_key
3. export AWS_DEFAULT_REGION=aws_region

```

Он настраивает переменную среды для AWS. Теперь выполним команду для развертывания Functionbeat:

```
./functionbeat -v -e -d "*" deploy cloudwatch
```

Другие параметры и вывод данных настраиваются так же, как и для прочих инструментов Beats.

Чтобы отправлять данные в облако Elastic Cloud, необходимо добавить код, указывающий идентификатор Cloud ID:

1. `cloud.id: "staging:dxMtZWFzdC0xLmF3er5mb3VuZC5reyRjZWM2ZjI2M-.WE3NGJnMjRjZTMzYmI40DExYjg0Mjk0ZiRjNmMyY2E2ZDA0MjI0OWFmMGNjN-2Q3YT110TYyNTc0Mw=="`
2. `cloud.auth: "elastic:your_password".`

Вывод Elastic Cloud также можно настроить с помощью такой команды:

```
functionbeat -e -E cloud.id=<cloud-id> -E cloud.auth=<cloud. auth>"
```

Чтобы отправлять данные в Elasticsearch, установленную на машине или в облаке, в файл конфигурации необходимо добавить следующий код:

1. `output.elasticsearch:`
2. `hosts: ["localhost:9200"].`

Чтобы отправлять данные в Logstash, в файл конфигурации следует добавить такой код:

1. `output.logstash:`
2. `hosts: ["127.0.0.1:5044"]`

Таким образом, результаты Functionbeat можно выводить в Elasticsearch, Elastic Cloud или Logstash.

ИНТЕГРАЦИЯ КОМПОНЕНТОВ ELASTIC STACK

Elastic Stack — это набор инструментов и технологий с открытым исходным кодом, которые работают вместе для решения разнообразных задач обработки и анализа данных. Основные компоненты Elastic Stack — Elasticsearch, Logstash, Kibana и Beats. Их интеграция обеспечивает полноценный сквозной пайплайн обработки данных.

Данные можно собирать и обрабатывать с помощью Logstash или Beats, индексировать и хранить в Elasticsearch, а затем визуализировать и анализировать с помощью Kibana. Такая бесшовная интеграция обеспечивает эффективный поиск данных, мониторинг в реальном времени и расширенные возможности

аналитики в рамках стека Elastic. Рассмотрим пример, в котором будем обрабатывать логи Apache с помощью Logstash и хранить их в Elasticsearch.

Получение логов Apache с помощью Logstash

С помощью Logstash можно получать данные из разных источников, таких как MySQL, MongoDB, CSV, файлы, Elasticsearch и т. д. Для этого необходимо использовать соответствующие плагины ввода Logstash. В этой книге мы не сможем рассказать обо всем, но на простом примере рассмотрим, как обрабатывать данные с помощью Logstash. Лог доступа Apache — это неструктурированные данные, и их нужно обработать, чтобы сделать пригодными для поиска. Вот запись в логе Apache:

```
1. 127.0.0.1 - - [29/Feb/2020:18:43:46 +0530] "GET /test/admin.php HTTP/1.1" 200 2159 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:72.0) Gecko/20100101 Firefox/72.0".
2. 127.0.0.1 - - [29/Feb/2020:18:43:47+0530] "GET/test/admin.php?file=default.css&version=4.2.5&driver=mysql HTTP/1.1" 200 2343 "http://localhost/test/admin.php" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:72.0) Gecko/20100101 Firefox/72.0".
3. 127.0.0.1 - - [29/Feb/2020:18:43:47 +0530] "GET /test/adminer.css HTTP/1.1" 200 4883 "http://localhost/test/admin.php" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:72.0) Gecko/20100101 Firefox/72.0"
```

Обработаем логи Apache с помощью Logstash. Для этого воспользуемся плагином Logstash получения входных данных из файлов и укажем путь к файлу лога Apache. Поскольку эти логи не структурированы, их необходимо преобразовать в структурированный формат, прежде чем записывать в Elasticsearch. Для фильтрации можно использовать шаблон grok, чтобы написать собственное регулярное выражение, или воспользоваться имеющимися шаблонами Logstash. В данном случае для обработки логов Apache мы используем шаблон COMBINEDAPACHELOG. Создадим файл конфигурации `apache_data.conf` Apache log и напишем следующий код:

```
1. input {
2.   file {
3.     path => "/var/log/apache2/access.log"
4.     type => "apache_access"
5.   }
6. }
7. filter {
8.   grok {
9.     match =>{ "message" => "%{COMBINEDAPACHELOG}" }
10.  }
11. }
12. output {
13.   elasticsearch {
```

```

14.     action => "index"
15.     hosts => ["127.0.0.1:9200"]
16.     index => "apache_logs"
17.     user => elastic_username
18.     password => your_password
19.   }
20. }
```

Файл создан в каталоге `logstash/conf.d`, и его можно запустить с помощью такой команды:

```
bin/logstash -f /etc/logstash/conf.d/apache_data.conf
```

После успешного выполнения предыдущей команды Logstash создаст индекс `apache_logs` в Elasticsearch. Проверить это можно, выполнив команду:

```
curl -XGET "http://localhost:9200/_cat/indices?v"
```

Она выведет список всех индексов Elasticsearch, и наличие индекса `apache_logs` означает, что Logstash успешно обработал логи Apache. Документы внутри индекса можно увидеть, выполнив следующую команду:

```
curl -XGET "http://localhost:9200/apache_logs/_search"
```

Получим такой документ:

```

1. {
2.     "_index" : "apache_logs",
3.     "_type" : "_doc",
4.     "_id" : "-097j2wBDCiVKm3nWGRa",
5.     "_score" : 1.0,
6.     "_source" : {
7.         "auth" : "-",
8.         "message" : "127.0.0.1 -- [29/Feb/2020:15:05:20 +0530] \\"GET /.
test/admin.php HTTP/1.1\\" 200 2146 \"-\\" \\"Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:68.0) Gecko/20100101 Firefox/68.0\",
9.         "httpversion" : "1.1",
10.        "request" : "/test/admin.php",
11.        "@timestamp" : "2020-02-29T09:35:21.051Z",
12.        "clientip" : "127.0.0.1",
13.        "referrer" : "\\"-\\"",
14.        "@version" : "1",
15.        "ответ" : "200",
16.        "ident" : "-",
17.        "verb" : "GET",
18.        "host" : "KELLGGNLPTP0305",
19.        "timestamp" : "29/Feb/2020:15:05:20 +0530",
20.        "bytes" : "2146",
21.        "type" : "apache_access",
22.        "path" : "/var/log/apache2/access.log",
```

```
23.           "agent" : "{\"Mozilla/5.0 (X11; Ubuntu; Linux x86_64;  
rv:68.0) Gecko/20100101 Firefox/68.0\""  
24.       }  
25. }
```

Таким образом происходит преобразование неструктурированных записей лога Apache в структурированные данные Elasticsearch, которые легко искать, анализировать и визуализировать в Kibana. Можно настроить Logstash на получение данных из других источников и отправку их другим получателям.

ЗАКЛЮЧЕНИЕ

В этой главе мы познакомились с Elastic Stack и его компонентами. Мы подробно разобрали такие инструменты, как Elasticsearch, Logstash, Kibana и Beats. Далее мы рассмотрели, как они совместно применяются в различных сценариях использования и как настроить Beats или Logstash для отправки данных в Elasticsearch, а затем, используя Kibana, создать визуализацию данных.

В следующей главе мы расскажем о том, как подготовить данные, объясним, почему это важно, и поможем понять, что такое анализаторы Elasticsearch.

ВОПРОСЫ

1. Что такое Elastic Stack?
2. Каковы компоненты стека Elastic Stack?
3. Что такое Logstash и зачем его использовать?
4. Что такое плагины Logstash и какие они бывают?
5. Создайте файл конфигурации Logstash и получите данные из файлов логов.
6. Настройте Filebeat на получение данных лога Apache.

ГЛАВА 4

Подготовка данных к индексированию

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели стек Elastic Stack и его компоненты, такие как Elasticsearch, Logstash, Kibana и Beats. Мы также обсудили, как настроить инструменты Beats для отправки данных в Logstash или Elasticsearch. В этой главе мы сосредоточимся на подготовке данных к индексированию. Мы разберем концепцию анализа данных и научимся определять подходящий способ анализа данных. Кроме того, мы рассмотрим различные типы анализаторов, нормализаторов, токенизаторов, фильтров токенов и фильтров символов, доступных в Elasticsearch.

СТРУКТУРА

В этой главе:

- Важность подготовки данных к индексированию
- Введение в анализаторы Elasticsearch
- Что такое токенизаторы в Elasticsearch
- Что такое фильтры токенов
- Что такое фильтры символов в Elasticsearch
- Что такое нормализаторы

ЦЕЛИ

Изучив эту главу, вы получите представление о значении подготовки данных в Elasticsearch. Вы поймете роль анализаторов Elasticsearch в эффективной обработке и индексации текстовых данных. Кроме того, вы изучите различные важные компоненты анализаторов, включая нормализаторы, токенизаторы, фильтры лексем и фильтры символов. Это понимание позволит вам оптимизировать обработку и индексирование текстовой информации в Elasticsearch для более эффективного и точного поиска.

ВАЖНОСТЬ ПОДГОТОВКИ ДАННЫХ ПЕРЕД ИНДЕКСИРОВАНИЕМ

Поскольку объем данных, которые мы храним, продолжает расти в геометрической прогрессии, извлекать из них нужную информацию становится все сложнее. Поиск подразумевает не только точные совпадения. Поисковые запросы могут отличаться от реальных сохраненных данных, например содержать опечатки, синонимы или фонетические вариации. Для таких случаев очень важно заранее спланировать стратегию индексирования. Следует учитывать возможные сценарии и определить уровень поддержки, необходимый для поиска данных, в том числе рассмотреть такие варианты, как нечеткий поиск, поиск по синонимам или фонетический поиск методом стемминга¹ поискового термина. Иногда важно не пропустить релевантные результаты, даже если пользователь ввел неверное слово или написал его с ошибками.

АНАЛИЗАТОРЫ ELASTICSEARCH

Анализаторы — это специальные алгоритмы, которые преобразуют значения строкового поля в термины и сохраняют в виде инвертированного индекса. Существуют различные типы анализаторов, каждый со своей особой логикой парсинга текста. Выбор подходящего анализатора для конкретного сценария использования — это целое искусство, поскольку для разных сценариев существуют свои анализаторы.

Анализатор Elasticsearch объединяет в себе фильтр символов, токенизатор и фильтр токенов. Фильтры символов предварительно обрабатывают текст, выполняя такие операции, как удаление HTML или замена определенных

¹ Стемминг — процесс поиска основы заданного слова, см. <https://ru.wikipedia.org/wiki/Стемминг>. — Примеч. ред.

символов. Затем токенизатор разбивает текст на отдельные токены на основе определенных правил, например пробельные символы или знаки препинания. Наконец, фильтры токенов изменяют токены, применяя различные преобразования: перевод в нижний регистр, стемминг, удаление стоп-слов и т. д.

Комбинируя эти компоненты, анализаторы обрабатывают и индексируют текстовые данные, обеспечивая эффективный поиск и извлечение информации. Выбор анализатора зависит от таких факторов, как язык, тип данных и особые требования задачи.

Компоненты анализатора представлены на рис. 4.1.

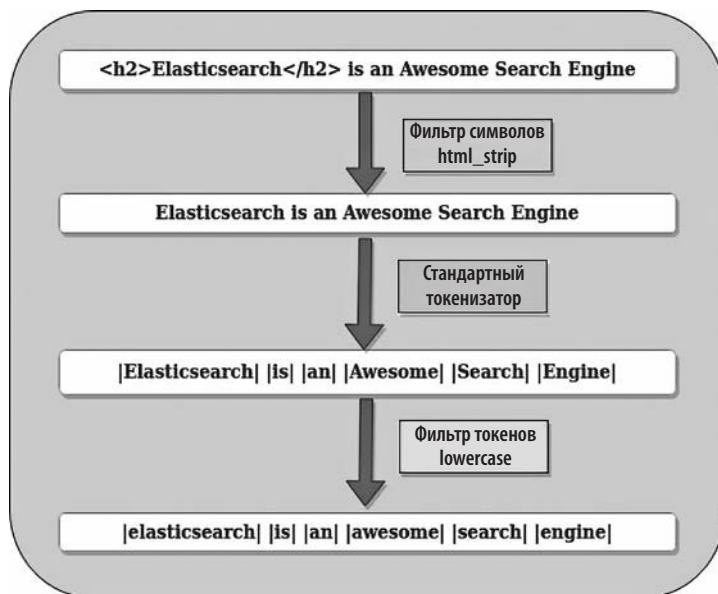


Рис. 4.1. Анализатор Elasticsearch

На рисунке показано, что задача анализатора выполняется благодаря совместной работе фильтра символов, токенизатора и фильтра токенов. На диаграмме мы видим текст *Elasticsearch is an Awesome Search Engine*¹, в котором изначально присутствуют теги *<h2>*. На первом этапе фильтр символов *html_strip* удаляет из текста теги *<h2>*. После этого в игру вступает стандартный токенизатор, который разбивает предложение на отдельные токены. Наконец, фильтр *lowercase* переводит символы в нижний регистр. В итоге получаем предложение, записанное строчными буквами.

¹ «Elasticsearch — потрясающий поисковый движок». — Примеч. ред.

По умолчанию с Elasticsearch поставляется множество анализаторов. Настроить пользовательские анализаторы можно с помощью API настроек Elasticsearch.

Вот пример настройки пользовательского анализатора с помощью API настроеек Elasticsearch:

```

1. PUT /index_name
2. {
3.   "settings": {
4.     "analysis": {
5.       "analyzer": {
6.         "custom_analyzer": {
7.           "type": "custom",
8.           "tokenizer": "standard",
9.           "filter": [
10.             "lowercase",
11.             "stop"
12.           ]
13.         }
14.       }
15.     }
16.   }
17. }
```

В приведенном выше фрагменте кода мы определяем пользовательский анализатор с именем "custom_analyzer". Он использует токенизатор "standard", который разбивает текст на отдельные токены с учетом пробелов и знаков препинания. Анализатор также включает два фильтра: "lowercase" и "stop". Фильтр "lowercase" преобразует токены в строчные, а фильтр "stop" удаляет распространенные стоп-слова, такие как *a*, *the*, *and* и т. д.

После того как пользовательский анализатор настроен, его можно связать с полем, указав его имя в сопоставлении или используя его в поисковом запросе.

Пользовательские анализаторы позволяют точно настроить процесс анализа текста в соответствии с особыми требованиями, такими как правила токенизации, характерные для конкретного языка, стемминг или стоп-слова в заданной предметной области. Пользовательские анализаторы обеспечивают больше контроля над индексированием и поиском текстовых данных в Elasticsearch.

Рассмотрим еще один пример. В следующем фрагменте кода показана настройка пользовательского анализатора:

```

1. PUT /index_name/_settings
2. {
3.   "index": {
4.     "analysis": {
5.       "analyzer": {
6.         "customHTMLSnowball": {
```

```

7.      "type": "custom",
8.      "char_filter": [
9.          "html_strip"
10.         ],
11.         "tokenizer": "standard",
12.         "filter": [
13.             "lowercase",
14.             "stop",
15.             "snowball"
16.         ]
17.     }
18.   }
19. }
20. }
21. }
```

Вот что происходит в этом фрагменте:

- используем фильтр символов `html_strip`, чтобы удалить все HTML-теги из исходного текста;
- разбиваем слова и удаляем знаки препинания с помощью стандартного токенизатора;
- затем применяем фильтры токенов, первый из которых — `lowercase`, чтобы преобразовать все токены в строчные символы;
- после этого используем фильтр стоп-слов, чтобы удалить все стоп-слова, такие как `the`, `and` и т. д.;
- наконец, применяем фильтр `snowball` («снежный ком») для стемминга всех токенов.

Итак, пользовательский анализатор в Elasticsearch можно настроить с помощью API-настроек. Но чтобы изменить анализатор, нужно сначала закрыть индекс, а для этого выполнить следующую команду:

```
POST /index_name/_close
```

После того как индекс закрыт, можно изменить анализатор и затем открыть индекс с помощью команды ниже:

```
POST /index_name/_open
```

Можно применить анализатор к любому полю, см. следующий код:

```

1. PUT /index_name
2. {
3.   "mappings": {
4.     "properties": {
5.       "text": {
```

```

6.      "type": "text",
7.      "fields": {
8.          "english": {
9.              "type": "text",
10.             "analyzer": "english"
11.         }
12.     }
13.   }
14. }
15. }
16. }
```

В этом коде для текстового поля используется стандартный анализатор по умолчанию. Мы применяем анализатор английского языка на уровне поля, выполняющий стемминг и удаляющий стоп-слова. API `_analyze` позволяет увидеть процесс анализа. Рассмотрим пример:

```

1. GET index_name/_analyze
2. {
3.   "field": "text",
4.   "text": "Elasticsearch is an awesome search engine".
5. }
```

Используем текстовое поле из запроса выше и получим следующий результат:

```

1. {
2.   "tokens" : [
3.     {
4.       "token" : "elasticsearch",
5.       "start_offset" : 0,
6.       "end_offset" : 13,
7.       "type" : "<ALPHANUM>",
8.       "position" : 0
9.     },
10.    {
11.      "token" : "is",
12.      "start_offset" : 14,
13.      "end_offset" : 16,
14.      "type" : "<ALPHANUM>",
15.      "position" : 1
16.    },
17.    {
18.      "token" : "an",
19.      "start_offset" : 17,
20.      "end_offset" : 19,
21.      "type" : "<ALPHANUM>",
22.      "position" : 2
23.    },
24.    {
25.      "token" : "awesome",
26.      "start_offset" : 20,
```

```

27.      "end_offset" : 27,
28.      "type" : "<ALPHANUM>",
29.      "position" : 3
30.    },
31.    {
32.      "token" : "search",
33.      "start_offset" : 28,
34.      "end_offset" : 34,
35.      "type" : "<ALPHANUM>",
36.      "position" : 4
37.    },
38.    {
39.      "token" : "engine",
40.      "start_offset" : 35,
41.      "end_offset" : 41,
42.      "type" : "<ALPHANUM>",
43.      "position" : 5
44.    }
45.  ]
46. }
```

Проверить, как работает анализатор для поля `text.english`, можно с помощью следующего кода:

```

1. GET index_name/_analyze
2. {
3.   "field": "text.english",
4.   "text": "Elasticsearch is an awesome search engine".
5. }
```

Получим такой результат:

```

1. {
2.   "tokens" : [
3.     {
4.       "token" : "elasticsearch",
5.       "start_offset" : 0,
6.       "end_offset" : 13,
7.       "тип" : "<ALPHANUM>",
8.       "position" : 0
9.     },
10.    {
11.      "token" : "awesom",
12.      "start_offset" : 20,
13.      "end_offset" : 27,
14.      "type" : "<ALPHANUM>",
15.      "position" : 3
16.    },
17.    {
18.      "token" : "search",
19.      "start_offset" : 28,
```

```

20.      "end_offset" : 34,
21.      "type" : "<ALPHANUM>",
22.      "position" : 4
23.    },
24.    {
25.      "token" : "engin",
26.      "start_offset" : 35,
27.      "end_offset" : 41,
28.      "type" : "<ALPHANUM>",
29.      "position" : 5
30.    }
31.  ]
32. }
```

Видим, что стоп-слова удалены, а токены стеммированы.

Встроенные анализаторы

Elasticsearch поставляется с набором встроенных анализаторов; их можно использовать напрямую, без дополнительной настройки. Рассмотрим некоторые из них подробнее.

- **Стандартный анализатор** — используется по умолчанию, если не указано иное. Он разбивает текст на токены с учетом пробелов и знаков препинания, преобразует токены к строчному виду и удаляет распространенные английские стоп-слова.
- **Простой анализатор** — разбивает текст на токены на основании небуквенных символов и преобразует токены к строчному виду.
- **Анализатор пробелов** — разбивает текст на токены на основе символов пробелов.
- **Анализатор ключевых слов** — рассматривает весь текст как один токен, без токенизации или преобразования к строчному виду.
- **Языковые анализаторы** — Elasticsearch предоставляет анализаторы для разных языков, таких как английский, испанский, французский, немецкий и др. Эти анализаторы проводят токенизацию, удаление стоп-слов и стемминг на основании правил языка.
- **Пользовательские анализаторы** — помимо встроенных анализаторов, Elasticsearch позволяет создавать собственные, комбинируя различные фильтры символов, токенизаторы и фильтры токенов в соответствии с требованиями проекта.

Каждый анализатор имеет свое назначение и выбирается в зависимости от типа данных и желаемого результата анализа. Выбрав подходящий анализатор, вы

сможете обеспечить индексацию и поиск текстовых данных в соответствии с потребностями своего приложения.

Стоит отметить, что анализаторы используются в процессе индексирования для преобразования текста, и они же обычно применяются для обеспечения совместимости в ходе поиска. Рассмотрим подробнее несколько анализаторов.

Стандартный анализатор

Стандартный анализатор (**Standard**) применяется по умолчанию, если не указан иной. Для токенизации предложений он использует алгоритм Unicode Text Segmentation, основанный на грамматике. В следующем фрагменте кода показан стандартный анализатор:

```
1. POST _analyze
2. {
3.   "analyzer": "standard",
4.   "text": "Elasticsearch is an awesome search engine"
5. }
```

Код выше вернет такие токены:

```
[elasticsearch, is, awesome, search, engine].
```

Код содержит следующие параметры:

- **max_token_length** — обозначает максимальную длину токена. Если ее указать, анализатор будет выделять токены с указанной длиной относительно значения **max_token_length**. По умолчанию значение **max_token_length** равно 255 символам.
- **stopwords** — позволяет указать стоп-слова или использовать **the_english** для указания заранее определенных стоп-слов. Также можно задать список стоп-слов в виде массива. По умолчанию список стоп-слов пуст (**default_none_stopword**).
- **stopword_path** — здесь можно указать путь к файлу, содержащему стоп-слова.

Таким образом, указанные выше параметры используются вместе с запросом **analyze** для настройки деталей процесса анализа.

Простой анализатор

Простой анализатор (**Simple**) создает токены по каждому небуквенному поисковому термину; это может быть пробел или любой другой символ. Простой анализатор настроить нельзя, так как он содержит только токенизатор **lowercase**.

В следующем фрагменте кода показан простой анализатор и то, как он выделяет токены:

```
1. POST _analyze
2. {
3.   "analyzer": "simple",
4.   "text": "Elasticsearch, is an awesome search engine!"
5. }
```

Код выше сгенерирует следующие токены:

```
[elasticsearch, is, awesome, search, engine].
```

Таким образом, он разобьет текст на токены, если встретит какой-либо специальный символ между словами.

Анализатор пробелов

Анализатор пробелов (`Whitespace`) преобразует текст в токены, используя пробельные символы. Анализатор пробелов не приводит токены к строчному виду:

```
1. POST _analyze
2. {
3.   "analyzer": "whitespace",
4.   "text": "Elasticsearch, is an awesome search engine!"
5. }
```

Приведенный выше код сгенерирует следующие токены:

```
[Elasticsearch, is, an, awesome, search, engine!]
```

Видим, что слово `Elasticsearch` не приведено к строчному виду.

Анализатор стоп-слов

Анализатор стоп-слов (`Stop`) очень похож на простой анализатор, но с дополнительной функцией удаления стоп-слов. См. следующий пример применения анализатора стоп-слов к уже знакомому нам тексту:

```
1. POST _analyze
2. {
3.   "analyzer": "stop",
4.   "text": "Elasticsearch, is an awesome search engine!".
5. }
```

Приведенный выше код сгенерирует следующие токены:

```
[elasticsearch, awesome, search, engine].
```

Видим, что стоп-анализатор удалил стоп-слова `is` и `a`.

Анализатор ключевых слов

Анализатор ключевых слов (**Keyword**) возвращает один токен из целой входной строки. В следующем примере показан принцип его работы:

```

1. POST _analyze
2. {
3.   "analyzer": "keyword",
4.   "text": "Elasticsearch, is an awesome search engine!".
5. }
```

Приведенный выше код генерирует следующие токены:

```
[Elasticsearch, is an awesome search engine!]
```

Таким образом, из всей входной строки генерируется один токен.

Анализатор шаблонов

Анализатор шаблонов (**Pattern**) разбивает текст на токены с помощью регулярного выражения. Он также поддерживает стоп-слова и преобразует токены к строчному виду.

Ниже приведен код создания анализатора шаблонов путем предоставления регулярного выражения, не содержащего слов:

```

1. PUT my_index2
2. {
3.   "settings": {
4.     "analysis": {
5.       "analyzer": {
6.         "my_email_analyzer": {
7.           "type": "pattern",
8.           "pattern": "\W|_",
9.           "lowercase": true
10.        }
11.      }
12.    }
13.  }
14. }
```

После создания анализатора его можно применить к строке. См. следующий код:

```

1. POST my_index2/_analyze
2. {
3.   "analyzer": "my_email_analyzer",
4.   "text": "sophia.julian@yopmail.com"
5. }
```

Этот код выведет такие токены:

```
[sophia, julian, yopmail, com].
```

Таким образом, можно создать шаблон для соответствия и преобразовать строку в токены.

Языковые анализаторы

Языковой анализатор (Language) создает токены для текста на определенном языке. Поддерживается множество языков, такие как арабский,ベンガル語, чешский, датский, голландский, английский, финский, французский, немецкий, греческий, хинди, шведский и др.

Применим анализатор английского языка в качестве пользовательского:

```

1. PUT /english_index
2. {
3.   "settings": {
4.     "analysis": {
5.       "filter": {
6.         "english_stop": {
7.           "type": "stop",
8.           "stopwords": "_english_"
9.         },
10.        "english_keywords": {
11.          "type": "keyword_marker",
12.          "keywords": ["example"].
13.        },
14.        "english_stemmer": {
15.          "type": "stemmer",
16.          "language": "english"
17.        },
18.        "english_possessive_stemmer": {
19.          "type": "stemmer",
20.          "language": "possessive_english"
21.        }
22.      },
23.      "analyzer": {
24.        "rebuilt_english": {
25.          "tokenizer": "standard",
26.          "filter": [
27.            "english_possessive_stemmer",
28.            "lowercase",
29.            "english_stop",
30.            "english_keywords",
31.            "english_stemmer"
32.          ]
33.        }
34.      }
}
```

```
35.      }
36.    }
37. }
```

Точно так же можно применить любой другой языковой анализатор.

Анализатор отпечатков

Для создания кластеров анализатор отпечатков (**Fingerprint**) использует алгоритм формирования отпечатков — фингерпринтинг. Он преобразует текст к строчному виду, удаляет символы расширенных наборов, дедуплицирует и объединяет слова в один токен. При составлении запроса с помощью анализатора отпечатков стоп-слова удаляются. Ниже приведен код анализатора:

```
1. POST _analyze
2. {
3.   "analyzer": "fingerprint",
4.   "text": "Elasticsearch, is an awesome search engine! awesome".
5. }
```

Этот код генерирует следующее ключевое слово:

```
[an awesome.elasticsearch.engine.is.search].
```

Это ключевое слово представляет собой один токен, который генерируется после дедупликации текста.

Пользовательские анализаторы

Если ни один из встроенных анализаторов не подходит, можно создать пользовательский. В пользовательском анализаторе комбинируют различные фильтры символов, токенизаторы и фильтры токенов и настраивают токенизатор и фильтры в соответствии с требованиями. В состав пользовательских анализаторов входят следующие элементы:

- токенизатор;
- 0 или больше фильтров символов;
- 0 или больше фильтров токенов.

ТОКЕНИЗАТОРЫ ELASTICSEARCH

Токенизаторы получают поток символов из строки и преобразуют его в отдельные слова, называемые **токенами**. Токенизаторы также отслеживают порядок токенов, учитывая смещение начального и конечного символов.

Словесно-ориентированные токенизаторы

Давайте обсудим токенизаторы, используемые для разбики целого текста на отдельные слова.

Стандартный токенизатор

Стандартный токенизатор (`standard`) использует алгоритм Unicode Text Segmentation для генерации токенов на основе грамматики. Он поддерживает различные языки. В следующем фрагменте кода показан стандартный токенизатор, который применяется к тексту:

```
1. POST _analyze
2. {
3.   "tokenizer": "standard",
4.   "text": "Elasticsearch, is an awesome search engine!".
5. }
```

Выполнив этот код, получим следующие токены:

```
[Elasticsearch, is, awesome, search, engine].
```

Видим, что в вывод включены все стоп-слова, и токены не приведены к строчному виду. Можно изменить `max_token_length`, чтобы задать максимальный размер каждого токена. По умолчанию значение `max_token_length` равно 255. Чтобы установить `max_token_length` равным 6 символам, настроим токенизатор таким образом:

```
1. PUT my_index
2. {
3.   "settings": {
4.     "analysis": {
5.       "analyzer": {
6.         "my_analyzer": {
7.           "tokenizer": "my_tokenizer".
8.         }
9.       },
10.      "tokenizer": {
11.        "my_tokenizer": {
12.          "type": "standard",
13.          "max_token_length": 6
14.        }
15.      }
16.    }
17.  }
18. }
```

Этот код создает пользовательский анализатор `my_analyzer`. Используем его для анализа текста. Код будет таким:

```

1. POST my_index/_analyze
2. {
3.   "analyzer": "my_analyzer",
4.   "text": "Elasticsearch, is an awesome search engine!".
5. }
```

В результате будут сгенерированы следующие токены:

`[Elasti, csearc, h, is, an awesom, e, search, engine].`

Таким образом, при использовании параметра `max_token_length` токены будут содержать не более 6 символов.

Буквенный токенизатор

Буквенный токенизатор (`letter`) преобразует текст в токен, когда встречает небуквенный символ.

В следующем фрагменте кода показан буквенный токенизатор:

```

1. POST _analyze
2. {
3.   "tokenizer": "letter",
4.   "text": " Elasticsearch, is an awesome search-engine!".
5. }
```

Выполнив этот код, получим следующие токены:

`[Elasticsearch, is, awesome, search, engine].`

Это результат применения буквенного токенизатора `letter`. Конфигурация этого токенизатора изменению не подлежит.

Токенизатор нижнего регистра

Токенизатор нижнего регистра (`lowercase`) похож на буквенный, поскольку он преобразует текст в токен, когда встречает любой небуквенный символ. Единственное отличие в том, что токенизатор нижнего регистра преобразует токены в нижний регистр. Возьмем тот же пример и применим на этот раз токенизатор нижнего регистра:

```

1. POST _analyze
2. {
3.   "tokenizer": "lowercase",
4.   "text": "Elasticsearch, is an awesome search-engine!".
5. }
```

Выполнив код выше, получим следующие токены:

`[elasticsearch, is, awesome, search, engine].`

Токенизатор пробелов

Токенизатор пробелов (`whitespace`) разбивает текст на токены по пробельным символам. В следующем фрагменте кода показано применение токенизатора `whitespace`:

```
1. POST _analyze
2. {
3.   "tokenizer": "whitespace",
4.   "text": " Elasticsearch, is an awesome search-engine!".
5. }
```

Выполнив код выше, получим следующие токены:

```
[Elasticsearch, is, an, awesome, search-engine!]
```

Видим, что токены выделены по пробельным символам, поэтому все остальные символы в них сохранены. Можно настроить токенизатор пробельных символов с помощью параметра `max_token_length`.

Токенизатор электронных адресов UAX URL

Токенизатор `uax_url_email` похож на стандартный токенизатор с одной дополнительной функцией: он распознает URL и адреса электронной почты и помещает их в один токен. В следующем фрагменте кода к заданному тексту применяется токенизатор `uax_url_email`:

```
1. POST _analyze
2. {
3.   "tokenizer": "uax_url_email",
4.   "text": "email me at sophia.julian@yopmail.com".
5. }
```

Выполнив код выше, получим следующие токены:

```
[Email, me, at, sophia.julian@yopmail.com].
```

Видим, что адрес электронной почты был преобразован в токен наряду с другими словами. Можно настроить токенизатор `uax_url_email` с помощью параметра `max_token_length`.

Классический токенизатор

Существует классический токенизатор (`classic`) для английского языка, основанный на грамматике. Он распознает адреса электронной почты, имена сетевых хостов и др. и сохраняет их в виде одного токена. Он выделяет слова на основе знаков препинания, удаляя их. Следующий фрагмент кода иллюстрирует работу токенизатора `classic`:

```

1. POST _analyze
2. {
3.   "tokenizer": "classic",
4.   "text": "Elasticsearch, is an awesome search-engine!".
5. }
```

Выполнив код выше, получим следующие токены:

[**Elasticsearch, is, awesome, search, engine**].

Видим, что дефис также учитывается при выделении токенов. Можно настроить классический токенизатор с помощью параметра `max_token_length`.

Токенизаторы частичных слов

Токенизаторы частичных слов (`partial word`) используются, если необходимо выполнить частичное сопоставление слов. Для этого слово разбивается на небольшие фрагменты.

Токенизатор N-Gram

Токенизатор N-Gram работает как скользящее окно, которое можно передвигать по слову. Он преобразует текст в непрерывную последовательность символов заданной длины. Эту последовательность можно использовать для частичного сопоставления слов, когда трудно подобрать точное совпадение. По умолчанию токенизатор `ngram` создает N-граммы, минимальная длина которых 1, максимальная — 2. Следующий фрагмент кода иллюстрирует пример работы токенизатора `ngram`:

```

1. POST _analyze
2. {
3.   "tokenizer": "ngram",
4.   "text": "Elasticsearch".
5. }
```

Получаем следующий результат:

[**E, E1, 1, 1a, a, as, s, st, t, ti, i, ic, c, cs, s, se, e, ea, a, ar, r, rc, c, ch, h**]

Видим, что слово `Elasticsearch` преобразуется в одно- и двухсимвольные токены. Токенизатор `ngram` принимает следующие параметры:

- `min_gram` — по умолчанию минимальная длина символа равна 1 и изменяется с помощью этого параметра;
- `max_gram` — по умолчанию максимальная длина символов равна 2 и изменяется с помощью этого параметра;
- `token_chars` — в этом параметре можно указать классы символов, которые нужно включить в токен;

- `custom_token_chars` — пользовательские символы, которые нужно включить в токен.

Параметры выше служат для дополнительной настройки токенизатора `ngram` и повышения его производительности.

Как настроить токенизатор n-gram для частичного поиска слов

Настройка значений `ngram` очень важна в Elasticsearch. Настраивая параметры `min_gram` и `max_gram`, можно управлять поведением `ngram`. Если для `min_gram` установить 1 или очень небольшое значение, будет сгенерировано много н-грамм и найдено много документов. Однако это не всегда хорошо, так как может привести к выдаче нерелевантных результатов поиска.

Например, если установить `min_gram` равным 1, это приведет к совпадению почти всех документов и отсутствию релевантности. Поэтому важно тщательно продумать значения `min_gram` и `max_gram`, чтобы найти баланс между релевантными и излишними совпадениями. Настроим токенизатор `ngram`, для которого `min_gram` равен 4, а `max_gram` 10:

```

1. PUT /sophia_index
2. {
3.   "settings": {
4.     "analysis": {
5.       "analyzer": {
6.         "autocomplete_analyzer": {
7.           "type": "custom",
8.           "tokenizer": "standard",
9.           "filter": [
10.             "lowercase",
11.             "autocomplete_ngram"
12.           ]
13.         }
14.       },
15.       "filter": {
16.         "autocomplete_ngram": {
17.           "type": "ngram",
18.           "min_gram": 4,
19.           "max_gram": 10
20.         }
21.       }
22.     }
23.   }
24. }
```

В этой конфигурации токенизатор `ngram` будет генерировать н-граммы длиной от 4 до 10 символов, обеспечивая баланс между релевантными и чрезмерными совпадениями. Настраивайте значения `min_gram` и `max_gram` в соответствии со своими конкретными требованиями.

ПРИМЕЧАНИЕ «`sophia_index`» необходимо заменить на фактическое имя индекса Elasticsearch, а значение «`title`» — на поле, которое требуется искать в индексе. Также настройте параметры запроса и названия полей в соответствии со своими требованиями.

Теперь проиндексируем несколько документов, чтобы можно было выполнить поиск. Для этого используем следующие запросы:

```

1. PUT /sophia_index/_doc/1
2. {
3.   "title": "Learning Elasticsearch 71".
4. }
5.
6. PUT /sophia_index/_doc/2
7. {
8.   "title": "Mastering Elasticsearch 62".
9. }
10.
11. PUT /sophia_index/_doc/3
12. {
13.   "title": "Elasticsearch 7 Quick Start Guide3".
14. }
```

С помощью приведенных выше запросов мы создали три документа в индексе `sophia_index`. Теперь попробуем выполнить поиск документов с помощью следующего запроса:

```

1. GET /sophia_index/_search
2. {
3.   "query": {
4.     "match_phrase_prefix": {
5.       "title": {
6.         "query": "elast",
7.         "slop": 0
8.       }
9.     }
10.   }
11. }
```

Этому запросу будут соответствовать все документы со строкой `elast`, поэтому получим следующие результаты:

```

1. {
2.   "took": 567,
3.   "timed_out": false,
4.   "_shards": {
```

¹ «Изучаем Elasticsearch 7». — Примеч. ред.

² «Подробное руководство по Elasticsearch 6». — Примеч. ред.

³ «Elasticsearch 7: краткий курс». — Примеч. ред.

```

5.     "total": 1,
6.     "successful": 1,
7.     "skipped": 0,
8.     "failed": 0
9.   },
10.  "hits": {
11.    "total": {
12.      "value": 3,
13.      "relation": "eq"
14.    },
15.    "max_score": 0.09141083,
16.    "hits": [
17.      {
18.        "_index": "sophia_index",
19.        "_id": "1",
20.        "_score": 0.09141083,
21.        "_source": {
22.          "title": "Learning Elasticsearch 7".
23.        }
24.      },
25.      {
26.        "_index": "sophia_index",
27.        "_id": "2",
28.        "_score": 0.09141083,
29.        "_source": {
30.          "title": "Mastering Elasticsearch 6".
31.        }
32.      },
33.      {
34.        "_index": "sophia_index",
35.        "_id": "3",
36.        "_score": 0.07296469,
37.        "_source": {
38.          "title": "Elasticsearch 7 Quick Start Guide".
39.        }
40.      }
41.    ]
42.  }
43. }
```

Таким образом, можно настроить `ngram` на основе требуемых критериев поиска. В запросе выше выполняется префиксный поиск, возвращающий все документы, начинающиеся со слова `elas`. Чтобы найти строку в любом месте текста, выполним следующий запрос:

```

1. GET /sophia_index/_search
2. {
3.   "query": {
4.     "wildcard": {
5.       "title": {
6.         "value": "*sea*"}
```

```

7.      }
8.    }
9.  }
10. }
```

С помощью этого запроса получим документы, в которых встречается слово *sea*, например *search*, *elasticsearch*, *sea* и т. д.

Токенизатор `edge_ngram`

Токенизатор `edge_ngram` похож на токенизатор `ngram`, с одним важным отличием: он выделяет заданную последовательность символов в начале каждого слова. Он хорошо подходит для запросов типа «поиск по вводу». Ниже приведен пример `edge_ngram`:

```

1. POST _analyze
2. {
3.   "tokenizer": "edge_ngram",
4.   "text": "Elasticsearch".
5. }
```

После выполнения этого кода получим такой результат:

`[E, E1]`

Для этого токена можно настроить следующие параметры:

- `min_gram` — по умолчанию минимальная длина символа равна 1 и изменяется с помощью этого параметра;
- `max_gram` — по умолчанию максимальная длина символов равна 2 и изменяется с помощью этого параметра;
- `token_chars` — в этом параметре можно указать классы символов, которые нужно включить в токен;
- `custom_token_chars` — пользовательские символы, которые нужно включить в токен.

Токенизаторы структурированного текста

Токенизаторы структурированного текста используются при работе со структурированными текстами, такими как адреса электронной почты, почтовые индексы, идентификаторы, пути и т. д.

Токенизатор ключевых слов

Токенизатор ключевых слов `keyword` принимает текст и выводит его в виде одного токена. В следующем фрагменте кода показан пример токенизатора `keyword`:

```

1. POST _analyze
2. {
3.   "tokenizer": "keyword",
4.   "text": "Elasticsearch".
5. }
```

Выполнив этот код, получим токен:

[Elasticsearch]

Видим, что одно слово `Elasticsearch` передается токенизатору ключевых слов и это же слово генерируется в качестве токена.

Токенизатор шаблонов

Токенизатор шаблонов разбивает текст на токены с помощью регулярного выражения. Он работает так же, как и анализатор шаблонов. По умолчанию используется шаблон `\W+`, который разбивает текст на токены, когда видит символы, не являющиеся словами.

Фильтры токенов

Фильтры токенов принимают потоки токенов от токенизатора и изменяют их. Эти модификации используются для преобразования текста в нижний регистр, удаления и добавления токенов и др. Существует множество встроенных фильтров, которые можно использовать для создания пользовательских анализаторов. Фильтры токенов выполняют следующие функции.

- Фильтр `lowercase` преобразует полученный от токенизатора текст в нижний регистр.
- Фильтр `uppercase` преобразует полученный от токенизатора текст в верхний регистр.
- Фильтр `stop` удаляет стоп-слова из потоков токенов.
- Фильтр `reverse` меняет токены на обратные.
- Фильтр `elision` служит для удаления элизий.
- Фильтр `truncate` сокращает токен до определенной длины. Длина по умолчанию устанавливается равной 10.
- Фильтр `unique` служит для индексирования уникальных токенов.
- Фильтр `duplicate` удаляет дубликаты токенов.

Таким образом, при необходимости выполнить какое-либо из указанных выше действий можно использовать соответствующий фильтр токенов.

ФИЛЬТРЫ СИМВОЛОВ В ELASTICSEARCH

Фильтры символов используются до передачи потока символов токенизатору. Они обрабатывают поток символов, удаляя, добавляя или изменяя их. В Elasticsearch существует множество встроенных фильтров символов; с их помощью можно создавать собственные анализаторы.

Фильтр HTML strip

С помощью фильтра `html_strip` можно удалить HTML-элементы из текста и заменить HTML-сущности, используя их декодированное значение. Следующий код иллюстрирует работу фильтра `html_strip`:

```
1. POST _analyze
2. {
3.   "tokenizer": "keyword",
4.   "char_filter": ["html_strip"],
5.   "text": "<p>I'm so <b>happy</b>!</p>"
6. }
```

Выполнив его, получим такое выражение¹:

```
[ \nI'm so happy!\n ]
```

Фильтр символов `html_strip` поддерживает параметр `escaped_tags`: в нем можно указать массив HTML-тегов, которые не требуется удалять из исходного текста.

Используя этот параметр, можно добавить HTML-теги, которые не нужно изменять.

Фильтр сопоставления

Фильтр сопоставления (mapping char) использует ассоциативный массив с ключами и их значениями. В случае совпадения текста с ключом фильтр заменяет ключ его значением. Используя этот фильтр символов, можно преобразовать язык в любой другой язык. Например, с его помощью легко преобразовать индийские символы в арабские и арабские в латинские.

Фильтр символов для замены шаблона

Используя фильтр символов `pattern_replace`, можно заменять символы, применяя регулярное выражение. В нем можно указать строку для замены. Фильтр символов `pattern_replace` поддерживает следующие параметры:

¹ «I'm so happy!» — «Я так счастлив!» — Примеч. ред.

- `pattern` — регулярное выражение в Java;
- `replacement` — строка замены, используемая для замены существующей строки при совпадении с регулярным выражением;
- `flags` — флаги, разделенные пайпом, для регулярного выражения Java.

Например, "CASE_INSENSITIVE|COMMENTS".

Перечисленные параметры используются для настройки поведения фильтра символов.

НОРМАЛИЗАТОРЫ

Нормализаторы очень похожи на анализаторы, но генерируют только один токен вместо нескольких. Нормализаторы не содержат токенизаторов и принимают некоторые фильтры символов и фильтры токенов. Доступные для использования фильтры: `asciifolding`, `cjk_width`, `decimal_digit`, `elision`, `lowercase` и `uppercase`. Наряду с этими фильтрами используются некоторые языковые фильтры. Можно создать пользовательский нормализатор, указав для него фильтры символов и фильтры токенов. См. следующий код:

```

1. PUT index
2. {
3.   "settings": {
4.     "analysis": {
5.       "char_filter": {
6.         "quote": {
7.           "type": "mapping",
8.           "mappings": [
9.             "" => """",
10.            "" => """
11.          ]
12.        }
13.      },
14.      "normalizer": {
15.        "my_normalizer": {
16.          "type": "custom",
17.          "char_filter": ["quote"],
18.          "filter": ["lowercase", "asciifolding"].
19.        }
20.      }
21.    }
22.  },
23.  "mappings": {
24.    "properties": {
25.      "foo": {
26.        "type": "keyword",
27.        "normalizer": "my_normalizer"
28.      }
29.    }
30.  }
31. }
```

```
28.      }
29.      }
30.    }
31. }
```

В приведенном примере пользовательский нормализатор использует фильтры символов `quote`, а также фильтры `lowercase` и `asciifolding`.

ЗАКЛЮЧЕНИЕ

В этой главе мы узнали, что такое анализ данных и почему он важен. Мы также узнали, что такое анализатор, нормализатор, токенизатор, фильтр токенов и фильтр символов. Затем мы рассмотрели различные типы анализаторов и области их применения, обсудили виды нормализаторов, токенизаторов, фильтров токенов и фильтров символов и области их применения.

В следующей главе мы рассмотрим получение, хранение и визуализацию данных. Мы также узнаем, как отправлять данные в Elasticsearch с помощью инструментов Beats. Кроме того, мы изучим, как настроить Logstash на получение данных из различных источников для отправки их в Elasticsearch.

ВОПРОСЫ

1. Что такое анализатор Elasticsearch?
2. Объясните работу встроенного анализатора.
3. Что такое анализатор шаблонов?
4. Для чего нужен токенизатор?
5. Для чего нужен фильтр токенов?
6. Для чего нужен фильтр символов?
7. Для чего нужны нормализаторы?

ГЛАВА 5

Импорт данных в Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели, как подготовить данные к индексации, узнали, что такое анализ данных и как выбрать подходящий тип анализа. Затем мы рассмотрели, какие типы анализаторов, нормализаторов, токенизаторов, фильтров токенов и фильтров символов существуют в Elasticsearch. Эту главу мы начнем с объяснения того, почему данные так важны для бизнеса. Далее обсудим, что такое получение, хранение и визуализация данных. После этого разберемся, как импортировать данные в Elasticsearch с помощью инструментов Beats. Далее вы узнаете, как использовать Logstash для импорта данных в Elasticsearch.

Итак, начнем с того, почему данные так важны.

СТРУКТУРА

В этой главе:

- Почему данные важны для бизнеса
- Доставка данных
- Поглощение данных
- Хранение данных
- Визуализация данных
- Импорт данных в Elasticsearch с помощью Beats

ЦЕЛИ

Прочитав эту главу, вы получите полное представление о вопросах, связанных с управлением данными, в том числе о ключевой роли данных

в бизнес-процессах и принятии решений. Вы также узнаете, какое значение имеют процессы доставки данных для обеспечения своевременного анализа и принятия обоснованных решений.

Вы разберетесь в фундаментальных концепциях, лежащих в основе получения, хранения и визуализации данных. Эти концепции важны для извлечения из данных значимых аналитических выводов и трендов. Кроме того, вы приобретете практические навыки использования двух важнейших инструментов — Elastic Beats и Logstash. Эти инструменты позволяют импортировать, обрабатывать и обогащать данные в целях еще более эффективного управления и использования их потенциала для роста и успеха бизнеса.

ПОЧЕМУ ДАННЫЕ ВАЖНЫ ДЛЯ БИЗНЕСА

Любому бизнесу крайне необходима актуальная информация, скрытая в необработанных данных. Чтобы получить полезную информацию, данные следует обработать. Типы данных могут быть разными, например временные ряды, структурированные или неструктурированные. Для извлечения полезной информации надо обеспечить единообразие данных. Часто данные поступают из разных источников. Чтобы установить взаимосвязь между данными, полученными из разных источников, необходимо их обработать и поместить на единый дашборд. Для того чтобы получить полную картину, надо сделать следующее:

- определить потребности бизнеса;
- определить все источники данных, необходимых для решения бизнес-задач;
- провести преобразование данных, чтобы обеспечить их единообразие; сюда относится преобразование неструктурированных данных в структурированные и изменение значений полей;
- получить данные и преобразовать их в соответствии с общей схемой;
- сохранить данные в одном месте;
- создать единый дашборд для визуализации и анализа данных.

Выполнив эти действия, мы соберем всю информацию, необходимую для понимания требований бизнеса. Когда данные будут собраны в едином формате, их можно будет проанализировать и сделать выводы, позволяющие провести преобразования в организации. Необработанные данные имеют особое значение для любого бизнеса. Организации по всему миру работают с данными, анализируя их и собирая из них полезную информацию. На основе этих данных они создают системы оповещений и тикетов для проактивного мониторинга. Кроме того, они применяют машинное обучение для выявления аномалий

и прогнозирования тенденций. Таким образом они готовятся к будущим вызовам и соответствующим образом улучшают рабочие процессы.

ДОСТАВКА ДАННЫХ

Данные доставляются с удаленных машин в централизованное место сбора. Архитектура доставки данных должна поддерживать транспортировку разных типов данных или событий, например структурированных или неструктурированных данных. Основная цель доставки — переместить данные в центральное хранилище для дальнейшего исследования. Доставка данных обычно осуществляется с помощью легковесных агентов, развертываемых для одной цели. Подобные доставщики устанавливаются и настраиваются для чтения данных на удаленных серверах и отправки их в центральное хранилище. При доставке данных необходимо учитывать следующее.

- Легковесные доставщики должны быть небольшими, чтобы не влиять на производительность реальных серверных процессов.
- Существует несколько инструментов доставки данных, которые поддерживают различные технологии. Некоторые из них привязаны к конкретным технологиям, а другие основаны на расширяемом фреймворке.
- Передача данных также требует обеспечения безопасности: данные с удаленного сервера необходимо отправлять по сквозному защищенному каналу.
- Важно понимать, как обрабатывается нагрузка. Управление нагрузкой должно обеспечивать стабильный прием данных в месте назначения. Кроме того, при возникновении проблем в месте назначения скорость доставки должна снижаться. Эта функция известна как механизм обратного давления (back-pressure).

Доставка данных — важная часть анализа данных, поскольку для анализа или визуализации необходимы данные из различных источников. Например, чтобы создать централизованную систему управления логами, надо отправлять в нее логи со всех серверов. Провести визуализацию или анализ данных можно будет только после того, как данные будут собраны в центральном хранилище.

ПОГЛОЩЕНИЕ ДАННЫХ

Поглощение данных (data ingestion) — это процесс, в котором участвуют различные транспортные протоколы, обычно используемые для разных форматов данных. Он обеспечивает возможность извлечения и преобразования данных

перед их хранением. Обработка данных — это процесс, в ходе которого данные извлекаются, преобразуются и загружаются. Он также известен как **извлечение, преобразование, загрузка (ETL – Extract, Transform, Load)**. ETL — это не что иное, как пайплайн ввода данных, с помощью которого можно получать данные с разных уровней доставки, например Beats или Logstash, и отправлять их на уровень хранения. Поглощение данных имеет ряд особенностей.

Основная цель уровня поглощения — подготовка данных, в ходе которой происходит парсинг и форматирование данных, а также их сопоставление с другими источниками, нормализация и обогащение перед хранением. Конечная цель — повысить качество данных, чтобы их можно было анализировать. Кроме того, улучшение данных позволяет избавиться от дополнительных накладных расходов на их обработку. Если данные не оформлены как следует, необходимо выполнить соответствующую предварительную обработку перед анализом. Например, данные могут различаться, если они получены из разных источников. Необходимо скорректировать различия, чтобы улучшить качество данных, провести анализ и создать визуализации.

- Уровень поглощения поддерживает архитектуру, использующую различные подключаемые модули, помогающие получать данные из разных источников и в разных местах назначения. Эти плагины можно использовать наряду с инструментом доставки для получения данных из конкретного места; например, для получения данных из файла, сети или базы данных используются соответствующие плагины, предназначенные для конкретных целей. Иногда доступно несколько вариантов получения данных. Выбор подходящего варианта зависит от конкретного сценария. Для получения определенных данных можно использовать и Beats, и Logstash. Если данные требуют преобразования, следует использовать Logstash, в противном случае подойдет Beats.
- Поглощение и преобразование данных требует большого количества вычислительных ресурсов. Поэтому для максимизации пропускной способности данных необходимо тщательно планировать ресурсы, например распределить нагрузку на несколько экземпляров получателей. Чтобы создать систему визуализации,ирующую (почти) в реальном времени, следует получать данные как можно быстрее. Для этого очень важно распределить нагрузку между получателями. Таким образом можно ускорить отправку данных на хранение, чтобы затем использовать их для визуализации.

Elastic Beats и Logstash используются для поглощения данных, поскольку их можно настроить на получение данных из любого источника. После получения данные преобразуются в соответствии с требованиями, а после преобразования отправляются на уровень хранения Elasticsearch. После сохранения в Elasticsearch настраивается уровень создания различных визуализаций

с использованием Kibana. Kibana предоставляет возможность создавать разные визуализации и интегрировать несколько визуализаций на дашборде.

ХРАНЕНИЕ ДАННЫХ

Сердцем архитектуры, основанной на данных, является хранение, поскольку данные можно анализировать только после того, как они будут сохранены. Хранение данных — это процесс, позволяющий хранить важные данные в течение длительного времени в соответствии с установленным периодом хранения. Хранение обеспечивает платформу для выполнения таких операций, как поиск, анализ и визуализация данных. В идеале система анализа данных занимает центральное место в системе хранения, поскольку данные поступают из различных источников и хранятся централизованно, а их анализ и визуализация проводятся из хранилища. Организуя хранение данных, необходимо учитывать следующее.

- Поскольку объем данных растет с каждым днем, можно столкнуться с проблемами, связанными с объемом хранилища; поэтому очень важно его планировать и учитывать такие параметры, как скорость индексирования данных, период хранения и т. д. Объемы данных могут быть разными, от гигабайт до терабайт и петабайт. Масштабируемость — важный параметр, поскольку при увеличении количества запросов и объема данных требуется горизонтальное масштабирование; оно обеспечивается путем добавления машин для плавного увеличения объема хранилища.
- Для хранения данных чаще всего используется высокораспределенное нереляционное хранилище. Его преимущество заключается в ускорении доступа к данным, поддержке разных типов данных и возможности анализировать большой объем данных. Данные можно разбить на разделы и разместить на разных машинах, чтобы распределить нагрузку и сбалансировать ее для чтения и записи.
- Уровень хранения также должен предоставлять API, чтобы другие уровни, например визуализации и анализа данных, могли использовать эти API для дальнейшей обработки. На уровне визуализации выполняется статистический анализ хранимых данных, а также их агрегация. Вся основная работа выполняется на уровне хранения, визуализация только использует данные и выводит результат.

Хранение — это основа обработки данных, и для организации хранения в основном используется Elasticsearch. Elasticsearch обеспечивает такие возможности, как доступность данных практически в реальном времени, поиск, агрегирование данных любого размера и шардирование для масштабирования кластеров Elasticsearch на нескольких узлах.

Функция шардирования в Elasticsearch позволяет горизонтально масштабировать Elasticsearch и легко управлять большими объемами данных. Поисковые возможности Elasticsearch делают его лучшим инструментом для хранения данных, поскольку иногда бывает сложно найти нужную информацию в огромном массиве данных. Функция агрегации данных Elasticsearch помогает быстро получить представление обо всем наборе данных и доступна любому пользователю.

На уровне визуализации Kibana также использует возможности агрегации данных Elasticsearch для создания графиков, диаграмм и пр. Elasticsearch также предоставляет REST API, позволяющие легко выполнять различные операции. Elasticsearch берет на себя всю тяжелую работу, так что уровню визуализации остается только визуализировать данные, не обрабатывая их.

ВИЗУАЛИЗАЦИЯ ДАННЫХ

Визуализация — важный уровень анализа данных, поскольку он обеспечивает их наглядное представление. Визуализация данных включает набор инструментов для построения графиков и диаграмм. Информативные и полезные графики и диаграммы можно добавлять на аналитические дашборды, которые помогут ответить на разнообразные вопросы, например, о том как работает система, все ли в порядке, каковы тренды и нет ли в системе проблем. Основная задача визуализации — собрать все важные KPI и вывести их на информативном дашборде. Используя этот дашборд, руководство сможет принимать взвешенные решения, касающиеся организации. Особенности уровня визуализации:

- Он должен быть легковесным и предоставлять возможности создания визуализаций и графиков на основе данных с уровня хранения. Все основные операции обработки должны выполняться на уровне хранения, а уровень визуализации должен только выводить результат.
- Он должен способствовать быстроте понимания данных, выводя все важные KPI, а также предоставлять возможность исследования данных.
- Он должен предоставлять наглядный ответ на вопросы, а не являться средством изучения всех имеющихся данных.
- Он должен выводить визуальные данные практически в реальном времени, чтобы у специалистов была возможность оперативно решить любую проблему.
- Фреймворк визуализации должен быть расширяемым, чтобы настраивать в нем существующие активы, удалять их или добавлять новые в соответствии с требованиями.

- Уровень визуализации должен предоставлять возможность внешнего доступа к дашборду, чтобы использовать его в любом внешнем приложении.
- Желательно, чтобы уровень визуализации предоставлял дополнительные функции «из коробки», например машинное обучение для прогнозирования тенденций или выявления аномалий в данных.
- Уровень визуализации также должен предоставлять интерфейс, позволяющий легко импортировать данные без использования других инструментов.

На рис. 5.1 представлены различные уровни и их взаимодействие:

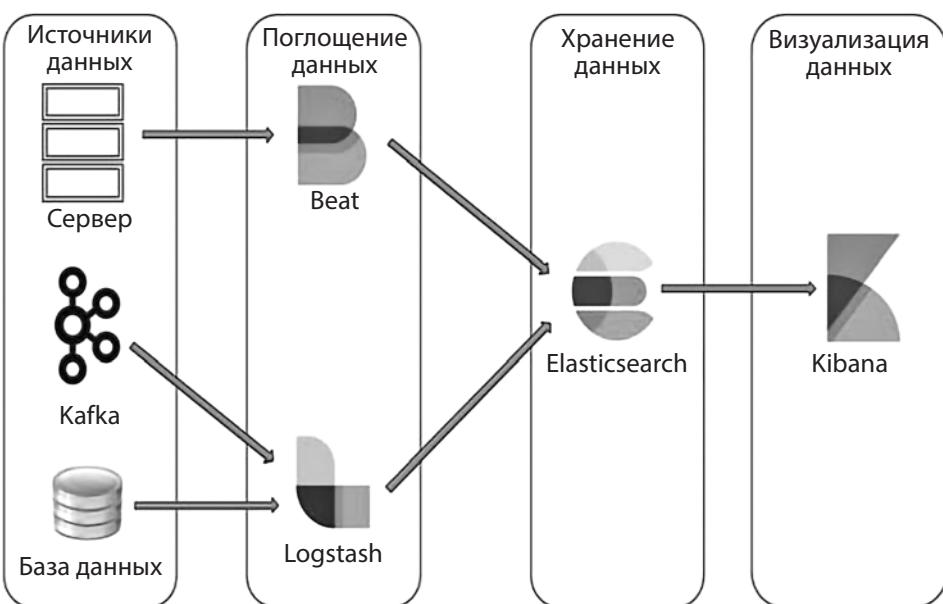


Рис. 5.1. Визуализация данных после их получения из различных источников

На рисунке показано, как данные поступают от источника на уровень визуализации через уровни поглощения и хранения. Источником данных может служить сервер, Kafka или любая база данных. Данные можно получать с помощью Beats или Logstash, а затем сохранять в Elasticsearch, основном инструменте хранения данных. Данные, хранящиеся в Elasticsearch, можно использовать для визуализации в Kibana.

ИМПОРТ ДАННЫХ В ELASTICSEARCH С ПОМОЩЬЮ BEATS

Elastic предоставляет легковесные инструменты Beats для доставки данных; они устанавливаются и настраиваются на удаленном сервере и используются для получения данных и их централизованного хранения. Рассмотрим, как настроить некоторые инструменты Beats для получения данных с удаленного сервера. На рис. 5.2 показана схема получения данных с помощью Beats.

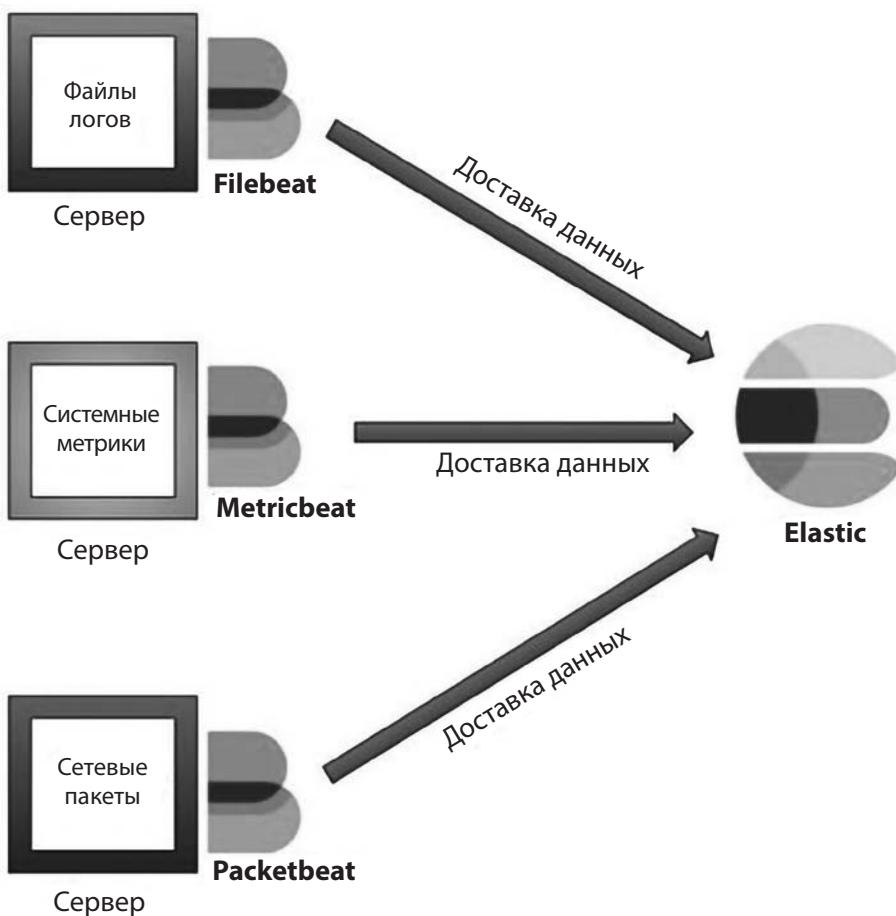


Рис. 5.2. Elastic Beats

На диаграмме показаны инструменты Filebeat, Metricbeat и Packetbeat, устанавливаемые и настраиваемые на серверах. Filebeat получает данные из файлов логов и отправляет их в централизованный кластер Elasticsearch. Metricbeat получает данные системных метрик с сервера и отправляет их в Elasticsearch, а Packetbeat получает данные сетевых пакетов и отправляет их в Elasticsearch. Теперь разберемся, как настроить эти инструменты для получения данных с удаленных серверов. Мы уже изучали Beats в предыдущей главе, поэтому рассмотрим, как импортировать данные с помощью разных видов Beats. Начнем с Filebeat.

Filebeat

Filebeat — это легковесный инструмент, с помощью которого можно отправлять логи в централизованное хранилище. Filebeat поддерживает внутренние модули для различных инструментов и программ, таких как Apache, NGINX и MySQL. Используя эти модули, Filebeat упрощает сбор, парсинг и визуализацию логов в форматах соответствующего модуля. Рассмотрим сценарий отправки логов Apache в Elasticsearch с помощью Filebeat. Первым шагом станет проверка правильности работы Filebeat. Проверим службу, выполнив следующую команду в Ubuntu:

```
sudo service filebeat status
```

Эта команда показывает состояние службы Filebeat; если служба не запущена, на терминале мы увидим следующий ответ:

В этом случае службу можно запустить, выполнив следующую команду:

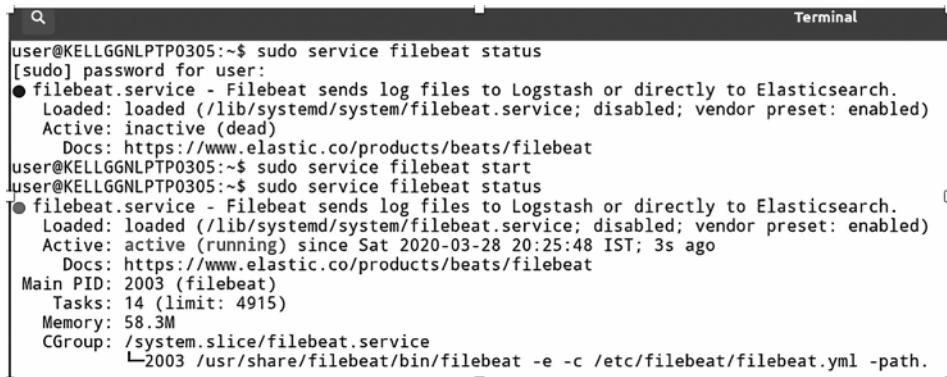
```
sudo service filebeat restart
```

Эта команда запустит службу Filebeat, и если после этого выполнить предыдущую команду status, получим следующий ответ:

1. filebeat.service - Filebeat sends log files to Logstash or directly to Elasticsearch
 2. Loaded: loaded (/lib/systemd/system/filebeat.service; disabled; vendor preset: enabled)
 3. Active: active (running) since Sat 2020-03-28 20:25:48 IST; 3s ago

¹ «Filebeat отправляет файлы логов в Logstash или напрямую в Elasticsearch». — Примеч. ред.

Этот ответ означает, что Filebeat запущен и работает. На рис. 5.3 показано состояние службы Filebeat.



```
user@KELLGGNLPTP0305:~$ sudo service filebeat status
[sudo] password for user:
● filebeat.service - Filebeat sends log files to Logstash or directly to Elasticsearch.
  Loaded: loaded (/lib/systemd/system/filebeat.service; disabled; vendor preset: enabled)
  Active: inactive (dead)
    Docs: https://www.elastic.co/products/beats/filebeat
user@KELLGGNLPTP0305:~$ sudo service filebeat start
user@KELLGGNLPTP0305:~$ sudo service filebeat status
● filebeat.service - Filebeat sends log files to Logstash or directly to Elasticsearch.
  Loaded: loaded (/lib/systemd/system/filebeat.service; disabled; vendor preset: enabled)
  Active: active (running) since Sat 2020-03-28 20:25:48 IST; 3s ago
    Docs: https://www.elastic.co/products/beats/filebeat
  Main PID: 2003 (filebeat)
    Tasks: 14 (limit: 4915)
   Memory: 58.3M
      CGroup: /system.slice/filebeat.service
              └─2003 /usr/share/filebeat/bin/filebeat -e -c /etc/filebeat/filebeat.yml -path.
```

Рис. 5.3. Запуск службы Filebeat

На рис. 5.3 показаны детали проверки состояния Filebeat. Первая команда проверяет статус и выводит ответ о том, что служба неактивна. Следующая команда запускает службу и выводит ответ о том, что служба активна.

На рисунке выше мы выполнили следующую команду, чтобы узнать состояние службы:

```
sudo service filebeat status
```

Модули Filebeat

Модули Filebeat упрощают настройку источников логов. Они позволяют Filebeat собирать, парсить и отправлять логи с различных сервисов или платформ без необходимости вручную настраивать каждый источник.

Перечислим несколько модулей, доступных в Filebeat:

- **Apache** — собирает и парсит логи HTTP-сервера Apache;
- **NGINX** — собирает и парсит логи сервера NGINX;
- **MySQL** — собирает и парсит логи сервера MySQL;
- **PostgreSQL** — собирает и парсит логи сервера PostgreSQL;
- **System** — собирает и парсит системные логи (например, syslog, systemd, логи событий Windows);
- **Redis** — собирает и парсит логи сервера Redis;
- **Docker** — собирает и парсит логи контейнеров Docker;

- **Kubernetes** — собирает и парсит логи контейнеров Kubernetes;
- **AWS** — отвечает за сбор логов из служб AWS, таких как CloudWatch, EC2 и S3;
- **AWS Fargate** — собирает логи из контейнеров AWS Fargate;
- **Google Cloud** — собирает логи из служб Google Cloud Platform (GCP), таких как Cloud Logging, Cloud Monitoring и Cloud Audit Logging;
- **Google Workspace** — собирает логи из рабочей среды Google (Gmail, Drive, Docs и т. д.);
- **Barracuda** — собирает логи с брандмауэров Barracuda и устройств защиты электронной почты;
- **Microsoft** — собирает логи систем и приложений Microsoft Windows;
- **NATS** — собирает логи системы обмена сообщениями NATS;
- **NetFlow** — собирает данные о сетевом трафике NetFlow.

Чтобы просмотреть доступные модули в Filebeat, используйте следующую команду:

```
filebeat modules list
```

Она отобразит список доступных модулей и их статус (включен или отключен).

Чтобы включить модуль, воспользуйтесь командой

```
filebeat modules enable <module_name>
```

Замените `<module_name>` на имя модуля, который необходимо включить. Например, чтобы включить модуль Apache, выполните команду

```
filebeat modules enable apache
```

После включения модуля Filebeat автоматически настроит для него необходимые входы, процессоры и выходы на основе заранее заданной конфигурации.

После включения или изменения модулей рекомендуется перезапустить Filebeat, чтобы применить изменения:

```
filebeat service filebeat restart
```

ПРИМЕЧАНИЕ Приведенные выше команды предполагают, что Filebeat установлен из официального пакета для вашей операционной системы. Если вы используете другой метод установки или настраиваемую конфигурацию, вам может потребоваться скорректировать команды соответствующим образом.

Получение логов Apache с помощью Filebeat

Теперь настроим Filebeat на передачу логов Apache с сервера в Elasticsearch, где эти данные будут храниться для дальнейшего анализа и визуализации. Ниже приведен формат записи лога доступа Apache 2:

```
1. 127.0.0.1 - - [14/Jun/2023:19:49:07 +0530] "GET /test.  
    html HTTP/1.1" 200 660 "-" "Mozilla/5.0 (X11; Ubuntu;  
    Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0".  
2. 127.0.0.1 - - [14/Jun/2023:19:49:07 +0530] "GET /favicon.  
    ico HTTP/1.1" 404 487 "http://localhost/test.html" "Mozilla/5.0  
    (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0".  
3. 127.0.0.1 - - [14/Jun/2023:19:49:26 +0530] "GET /test.  
    html HTTP/1.1" 200 599 "-" "Mozilla/5.0 (X11; Ubuntu;  
    Linux x86_64; rv:109.0)  
    Gecko/20100101 Firefox/110.0"  
4. 127.0.0.1 - - [14/Jun/2023:19:49:54 +0530] "GET /test.  
    html HTTP/1.1" 200 614 "-" "Mozilla/5.0 (X11; Ubuntu;  
    Linux x86_64; rv:109.0)  
    Gecko/20100101 Firefox/110.0"
```

Чтобы настроить Filebeat на чтение данных логов доступа Apache 2, сначала нужно внести следующие изменения в файл конфигурации `filebeat.yml`:

```
1. # ===== Filebeat inputs =====  
2. filebeat.inputs:  
3.  
4. # Большинство параметров устанавливается на уровне ввода, поэтому  
5. # для разных типов ввода можно настраивать разные конфигурации.  
6. # Ниже приведены конфигурации для конкретных видов ввода.  
7.  
8. # filestream – это тип входного потока для сбора сообщений логов из файлов.  
9. - type: filestream  
10.  
11. # Уникальный идентификатор ввода, является обязательным.  
12. id: my-filestream-id  
13.  
14. # Измените значение на true, чтобы активировать эту конфигурацию.  
15. enabled: false  
16.  
17. # Пути, которые должны быть проидены. Устанавливаются на основе  
    # глобальных данных.  
18. paths:  
19.   - /var/log/apache2/*.log
```

В блоке «Filebeat inputs» необходимо указать путь к месту хранения лога. Поскольку задача — читать логи Apache 2, необходимо указать путь к местоположению лога Apache 2. Кроме того, мы указали `*.log`, чтобы Filebeat мог

читать все файлы логов в расположении `/var/log/apache2/`. Можно настроить другие расположения аналогичным образом или указать путь `/var/log/*.log`, чтобы читать все файлы лога в папке. Тогда Filebeat получит доступ ко всем файлам лога в папке `/var/log/`.

После установки местоположения лога нужно настроить вывод Elasticsearch, куда мы будем отправлять данные; для этого необходимо выполнить следующее:

```

1. ===== Outputs =====
2. # Настройте вывод для отправки собранных данных.
3.
4. # -----Вывод Elasticsearch -----
5. output.elasticsearch:
6.   # Массив хостов для подключения.
7.   hosts: ["localhost:9200"].
8.
9.   # Необязательный протокол и основные учетные данные.
10.  #protocol: "https"
11.  username: "elastic"
12.  password: "your_password"
```

В этом блоке файла `filebeat.yml` мы настраиваем вывод Elasticsearch. Здесь нужно указать адрес хоста, протокол, имя пользователя и пароль Elasticsearch, чтобы Filebeat мог установить соединение с Elasticsearch для отправки данных лога. Чтобы отправлять данные в Logstash, а не напрямую в Elasticsearch, необходимо настроить блок *вывода Logstash*, указав в нем данные о хосте Logstash и т. д.

Чтобы отправлять данные лога в Elastic Cloud, необходимо указать идентификатор облака в разделе *Elastic Cloud* файла `filebeat.yml`. Начиная с версии Beats 6.0.0, дашборды можно загружать через API Kibana. Для загрузки дашбордов Kibana необходимо настроить конечную точку Kibana в файле конфигурации Filebeat в разделе Kibana.

После внесения всех изменений необходимо запустить службу Filebeat. Теперь можно перечислить все индексы Elasticsearch, чтобы проверить, создан ли индекс Filebeat. Для этого нужно выполнить следующую команду:

```
curl -XGET "http://localhost:9200/_cat/indices?v"
```

Эта команда выведет список всех индексов, в котором можно увидеть индекс `filebeat-7.6.1`. Можно переписать имя индекса, указав его в конфигурации Filebeat. Теперь выведем список документов индекса, выполнив следующую команду:

```
curl -XGET "http://localhost:9200/.ds-file-
beat-8.8.1-2023.06.14-000001/_search?q=apache"
```

Эта команда открывает индекс, выводя список всех документов в нем. Формат документов показан ниже:

```
1. {
2.     "_index": ".ds-filebeat-8.8.1-2023.06.14-000001",
3.     "_id": "HpDLuogBr5DPUDndRi0J",
4.     "_score": 0.0072727595,
5.     "_source": {
6.         "agent": {
7.             "name": "anurag-3570R-370R-470R-450R-510R-4450RV",
8.             "id": "9bd9921b-7288-48a9-9c2a-0edec044fab8",
9.             "ephemeral_id": "34977feb-456b-4a1d-8795-3861b08e81f2",
10.            "type": "filebeat",
11.            "version": "8.8.1"
12.        },
13.        "log": {
14.            "file": {
15.                "path": "/var/log/apache2/access.log"
16.            },
17.            "offset": 4984
18.        },
19.        -----
20.        -----
21.        "event": {
22.            "ingested": "2023-06-14T16:44:23.844652681Z",
23.            "original": "127.0.0.1 - - [14/Jun/2023:22:14:15 +0530] \"GET /.
test.htmls HTTP/1.1\" 404 488 \"-\" \"Mozilla/5.0 (X11; Ubuntu; 25.
Linux x86_64; rv:109.0) Gecko/20100101 Firefox/110.0\",
24.            "kind": "event",
25.            "created": "2023-06-14T16:44:22.811Z",
26.            "module": "apache",
27.            "category": "web",
28.            "dataset": "apache.access",
29.            "outcome": "failure"
30.        },
31.        -----
32.        -----
33.    }
34. }
```

Настройка Filebeat позволит эффективно извлекать и индексировать логи в Elasticsearch, обеспечивая бесшовную интеграцию между источниками логов и кластером Elasticsearch. Хотя модули Filebeat совершенствуются, чтобы получать данные из все большего количества источников, таких как MySQL, NGINX и др., в центре внимания этой книги в первую очередь ключевые функции, связанные с Elastic Stack. Изучив эти функции, читатели получат полное представление об их значении для оптимизации работы Elasticsearch и использования всех ее возможностей.

Изменение имени индекса в Filebeat

Чтобы изменить имя индекса в Filebeat на Ubuntu, необходимо отредактировать файл конфигурации Filebeat. Вот что нужно для этого сделать:

1. Откройте файл конфигурации Filebeat в текстовом редакторе. По умолчанию файл конфигурации расположен по адресу `/etc/filebeat/filebeat.yml`.
2. Найдите в файле конфигурации блок `output.elasticsearch`. Это блок настроек вывода Elasticsearch.
3. В блоке `output.elasticsearch` добавьте или измените параметр `index`, чтобы задать желаемое имя индекса. Например, если вы хотите изменить имя индекса на `"mylogs"`, используйте следующую конфигурацию:

```
output.elasticsearch:
  1. hosts: ["localhost:9200"].
  2. index: "mylogs-%{+yyyy.MM.dd}"
```

4. В приведенном выше примере `%{+yyyy.MM.dd}` — это спецификатор формата даты, который добавляет текущую дату к имени индекса. Эта возможность позволяет создавать ежедневные индексы.
5. Сохраните изменения в файле конфигурации.
6. Перезапустите Filebeat, чтобы применить изменения конфигурации:

```
sudo service filebeat restart
```

После перезапуска Filebeat будет отправлять логи в новое указанное имя индекса в Elasticsearch. Убедитесь, что все соответствующие шаблоны индексов в Kibana или других инструментах, использующих индексы Elasticsearch, обновлены в соответствии с новым именем индекса.

Metricbeat

Metricbeat — это легковесный инструмент отправки данных, который настраивается на удаленном сервере для получения системных метрик, таких как объем использования процессора и памяти, дисковые операции, сетевые операции и метрики файловой системы. Кроме того, с помощью Metricbeat можно получать статистику процессов, запущенных на сервере.

Модули Metricbeat

Модули Metricbeat — это предварительно настроенные компоненты, которые упрощают сбор и отправку метрик из различных инструментов и сервисов. Каждый модуль предназначен для мониторинга определенной системы, приложения или сервиса и содержит предварительно заданные наборы метрик для сбора соответствующих метрик и данных.

Преимущества модулей Metricbeat:

- **Упрощенная конфигурация.** Модули поставляются с предварительно заданными конфигурациями, что упрощает настройку Metricbeat для сбора метрик из различных источников без необходимости вручную указывать каждый набор метрик и связанную с ним конфигурацию.
- **Стандартизованный сбор метрик.** Модули Metricbeat следуют лучшим практикам инструмента или сервиса, которые они отслеживают. Они содержат согласованные наборы метрик и сопоставления полей, обеспечивая совместимость и простую интеграцию с Elasticsearch и другими инструментами Elastic Stack.
- **Настройка и гибкость.** Хотя модули предоставляют конфигурации по умолчанию, их можно настроить в соответствии с конкретными требованиями мониторинга. Можно изменять наборы метрик, добавлять или удалять поля, настраивать интервалы сбора данных, а также включать или отключать определенные наборы метрик в модуле.
- **Расширяемость.** Metricbeat позволяет создавать собственные модули, если встроенные не охватывают конкретный сценарий использования. Такая гибкость позволяет отслеживать с помощью Metricbeat пользовательские приложения или сервисы.

После включения и настройки модулей Metricbeat автоматически собирает метрики и данные с поддерживаемых инструментов и сервисов, что делает его эффективным и удобным решением для мониторинга систем инфраструктуры.

Вот некоторые из доступных модулей Metricbeat:

- **System** — мониторинг метрик системного уровня, таких как объем использования процессора, памяти, диска, сетевая статистика и т. д.;
- **Apache** — собирает метрики и логи Apache HTTP Server, включая количество запросов в секунду, коды ответов, переданные байты и логи ошибок;
- **Nginx** — мониторинг метрик веб-сервера Nginx, включая активные соединения, статистику запросов/ответов, ошибки HTTP и логи доступа;
- **MySQL** — собирает метрики производительности базы данных MySQL, статистику запросов и информацию о состоянии;
- **PostgreSQL** — собирает метрики базы данных PostgreSQL, включая статистику соединений, использование буфера и памяти, скорость транзакций и многое другое;
- **MongoDB** — мониторит показатели базы данных MongoDB, включая операции, объем использования памяти, состояние наборов реплик и многое другое;

- **Elasticsearch** — получает метрики кластера Elasticsearch, включая состояние узлов, статистику индексов, производительность поиска и индексирования;
- **Docker** — собирает метрики Docker на уровне контейнеров и образов, включая данные об использовании процессора и памяти, сетевую статистику и логи контейнеров;
- **AWS** — собирает метрики сервисов AWS, таких как EC2, ECS и RDS;
- **Azure** — собирает метрики сервисов Azure, таких как Azure VM, Azure App Service и Azure Storage;
- **ActiveMQ** — собирает метрики брокеров сообщений ActiveMQ;
- **Kubernetes** — собирает метрики кластеров Kubernetes;
- **Prometheus** — собирает метрики системы мониторинга Prometheus;
- **Redis** — собирает метрики хранилища данных памяти Redis;
- **Solr** — собирает метрики поисковой системы Solr;
- **ZooKeeper** — собирает метрики сервиса распределенной координации ZooKeeper.

Это лишь несколько примеров доступных модулей в Metricbeat. Каждый модуль имеет свои параметры настройки и может быть включен или отключен в зависимости от требований к мониторингу. Чтобы включить определенный модуль, используйте команду `Metricbeat modules enable`, за которой следует название модуля. Например, чтобы включить модуль Apache, нужно выполнить команду:

```
metricbeat modules enable apache
```

ПРИМЕЧАНИЕ Перед запуском Metricbeat не забудьте настроить параметры конфигурации для каждого модуля в соответствии с конкретной средой.

Получение метрик сервера с помощью Metricbeat

Чтобы получить системные метрики с помощью Metricbeat, необходимо установить и настроить Metricbeat на сервере. Если вы еще не установили Metricbeat, обратитесь к главе 3 «Elastic Stack: экосистема Elasticsearch» за подробными инструкциями по установке. После установки Metricbeat можно приступать к настройке. Файл конфигурации Metricbeat — `metricbeat.yml`, доступный в каталоге установки.

В файле конфигурации вы найдете блоки и опции для настройки поведения Metricbeat. Следующий фрагмент кода показывает конфигурацию модуля для настройки директории, из которой можно читать файл конфигурации модуля:

```
1. #===== Modules configuration =====
2. metricbeat.config.modules:
3.   # Глобальный шаблон для загрузки конфигурации
4.   path: ${path.config}/modules.d/*.yml
5.
6.   # Установите значение true, чтобы активировать перезагрузку конфигурации
7. reload.enabled: false
8.
9. # Период, в течение которого файлы заданного пути проверяются на наличие
# изменений
10. #reload.period: 10 s
```

По умолчанию конфигурация берется из каталога `modules.d`, но можно изменить местоположение, отредактировав путь к каталогу в файле конфигурации `metricbeat.yml`.

После настройки модуля нужно настроить блок вывода Elasticsearch. В нем нужно настроить учетные данные Elasticsearch для отправки данных метрик в Elasticsearch. В следующем фрагменте кода показан блок вывода Elasticsearch файла конфигурации `Metricbeat`:

```
1. ----- Elasticsearch output -----
2. output.elasticsearch:
3.   # Массив хостов для подключения.
4.   hosts: ["localhost:9200"].
5.
6.   # Необязательный протокол и основные учетные данные.
7.   #protocol: "https"
8.   username: "elastic"
9.   password: "your_password"
```

Можно настроить вывод Elasticsearch для отправки данных Metricbeat в Elasticsearch. В конфигурации выше для этого необходимо указать протокол, имя пользователя, пароль и данные хоста. После настройки файла `metricbeat.yml` необходимо перезапустить службу Metricbeat с помощью следующей команды:

```
sudo service metricbeat restart
```

После запуска Metricbeat получит метрики сервера и отправит их в кластер Elasticsearch.

Просмотреть документы в индексе Metricbeat можно, выполнив команду

```
curl -XGET "http://localhost:9200/.ds-metric-
beat-8.8.1-2023.06.14-000001/_search"
```

Эта команда откроет индекс и список всех документов в нем. Формат документов следующий:

```
1. {
2.     "_index": ".ds-metricbeat-8.8.1-2023.06.14-000001",
3.     "_id": "tpAOu4gBr5DPUDnd3Dke",
4.     "_score": 1,
5.     "_source": {
6.         "@timestamp": "2023-06-14T17:58:12.275Z",
7.         "agent": {
8.             "name": "anurag-3570R-370R-470R-450R-510R-4450RV",
9.             "type": "metricbeat",
10.            "version": "8.8.1",
11.            "ephemeral_id": "73ad803c-9b0b-4758-93b1-49b8ab0b7231",
12.            "id": "ff218027-0c49-4a35-a4ba-cd5b25bca1e4"
13.        },
14.        "service": {
15.            "type": "system"
16.        },
17.        "system": {
18.            "cpu": {
19.                "total": {
20.                    "norm": {
21.                        "pct": 0.8299
22.                    },
23.                        "pct": 3.3198
24.                    },
25.                    "softirq": {
26.                        "pct": 0.001,
27.                        "norm": {
28.                            "pct": 0.0003
29.                        }
30.                    },
31.                    "system": {
32.                        "pct": 0.1673,
33.                        "norm": {
34.                            "pct": 0.0418
35.                        }
36.                    },
37.                    "cores": 4,
38.                    "idle": {
39.                        "pct": 0.65,
40.                        "norm": {
41.                            "pct": 0.1625
42.                        }
43.                    },
44.                    "user": {
45.                        "norm": {
46.                            "pct": 0.0746
47.                        },
48.                            "pct": 0.2983
49.                        }
50.                    }
```

```
51.      },
52.      "host": {
53.        "os": {
54.          "version": "22.04.2 LTS (Jammy Jellyfish)",
55.          "family": "debian",
56.          "name": "Ubuntu",
57.          "kernel": "5.19.0-43-generic",
58.          "codename": "jammy",
59.          "type": "linux",
60.          "platform": "ubuntu"
61.        },
62.        "cpu": {
63.          "usage": 0.8299
64.        },
65.      },
66.    }
```

Индекс Metricbeat содержит различные метрики сервера, такие как данные о файловой системе, ОС, процессоре, памяти и т. д. Таким образом, можно настроить Metricbeat на чтение системных метрик и индексирование их в Elasticsearch. Системные метрики включают в себя различные данные, такие как данные о файловой системе, ОС, ЦП, памяти и пр. Также через Metricbeat можно предварительно настроить дашборд Kibana.

Packetbeat

Packetbeat – это легковесный анализатор сетевых пакетов, который перехватывает и анализирует сетевой трафик в реальном времени. Он является частью стека Elastic Stack и предназначен для получения информации о поведении и производительности сети.

Packetbeat работает, перехватывая сетевой трафик и анализируя полученные пакеты. Он изучает содержимое каждого пакета и извлекает необходимую информацию, такую как IP-адреса источника и получателя, номера портов, протоколы и данные полезной нагрузки. Packetbeat поддерживает различные протоколы. Модули, входящие в Packetbeat:

- **AMQP** – собирает сетевой трафик AMQP;
- **DNS** – собирает сетевой трафик DNS;
- **Elasticsearch** – собирает сетевой трафик к кластеру Elasticsearch и от него;
- **ICMP** – собирает сетевой трафик ICMP;
- **Kafka** – собирает сетевой трафик Kafka;
- **Kubernetes** – собирает сетевой трафик к кластеру Kubernetes и из него;
- **MongoDB** – собирает сетевой трафик MongoDB;

- **MySQL** – собирает сетевой трафик MySQL;
- **Nginx** – собирает сетевой трафик, идущий к веб-серверам Nginx и от них;
- **OpenFlow** – собирает сетевой трафик OpenFlow;
- **PostgreSQL** – собирает сетевой трафик PostgreSQL;
- **Redis** – собирает сетевой трафик Redis;
- **SSL** – собирает сетевой трафик SSL;
- **TCP** – собирает сетевой трафик TCP;
- **TLS** – собирает сетевой трафик TLS;
- **UDP** – собирает сетевой трафик UDP;
- **VMware** – собирает сетевой трафик в среду VMware vSphere и из нее.

С помощью этих протоколов можно отслеживать определенный сетевой трафик. Packetbeat можно использовать для различных целей, в том числе:

- **Мониторинг сети.** Анализируя сетевой трафик, Packetbeat может обеспечивать видимость производительности сети, выявлять узкие места и обнаруживать аномалии или проблемы, связанные с сетью.
- **Анализ безопасности.** Packetbeat можно использовать для мониторинга и анализа безопасности. Он может обнаруживать и предупреждать о подозрительной сетевой активности, такой как необычные схемы трафика или потенциальные угрозы безопасности.
- **Мониторинг производительности приложений (APM).** Благодаря возможностям анализа специфических протоколов Packetbeat может контролировать и измерять производительность приложений и служб. Он может предоставить информацию о времени отклика, количестве ошибок и других показателях производительности.
- **Устранение неполадок.** Packetbeat может помочь в устраниении проблем, связанных с сетью или приложениями, путем захвата и анализа данных на уровне пакетов. Это поможет выявить первопричину проблем и получить аналитическую информацию для их решения.

Packetbeat легко интегрируется с другими компонентами стека Elastic Stack, такими как Elasticsearch для хранения и индексации перехваченных пакетных данных, Kibana для визуализации и анализа данных, а также Logstash для преобразования и обогащения данных, если это необходимо.

Используя Packetbeat, сетевые администраторы, команды обеспечения безопасности и разработчики могут получать полезную аналитическую информацию о сетевом трафике и приложениях в целях эффективного контроля, защиты и оптимизации систем.

Получение сетевых данных с помощью Packetbeat

Этапы установки и настройки Packetbeat были подробно описаны в главе 3 «Elastic Stack: экосистема Elasticsearch». Кратко напомним о них:

- Настройте сетевой интерфейс:** в файле конфигурации Packetbeat укажите сетевой интерфейс, с которого будет осуществляться перехват сетевого трафика.
- Определите протоколы и порты:** укажите в файле конфигурации Packetbeat типы протоколов и номера портов, с которых будет перехватываться сетевой трафик. Это необходимо, чтобы Packetbeat перехватывал только нужные данные.
- Настройте место назначения вывода:** в файле конфигурации Packetbeat укажите раздел вывода, куда отправлять захваченные данные. Это может быть Logstash, Elasticsearch Cloud, Elasticsearch Cluster или любое другое место назначения, которое вы выберете.
- Настройте дашборд Packetbeat:** чтобы создать стандартный дашборд Packetbeat для визуализации перехваченных данных, укажите конечную точку Kibana в файле конфигурации Packetbeat. Это обеспечит бесшовную интеграцию с Kibana для визуализации и анализа.

Чтобы просмотреть все доступные сетевые интерфейсы, выполните следующую команду:

```
packetbeat devices
```

Эта команда предоставит список сетевых интерфейсов, доступных для настройки в файле конфигурации Packetbeat. Получим такой вывод:

```
user@KELLGGNLPTP0305:/var/log$ packetbeat devices
0: wlo1 (No description available) (192.168.43.127 fe80::d8a9:723a:899d:eff7)
1: any (Pseudo-device that captures on all interfaces) (Not assigned ip address)
2: lo (No description available) (127.0.0.1 ::1)
3: virbr0 (No description available) (192.168.122.1)
4: docker0 (No description available) (172.17.0.1)
5: enp0s25 (No description available) (Not assigned ip address)
6: virbr0-nic (No description available) (Not assigned ip address)
7: nflog (Linux netfilter log (NFLOG) interface) (Not assigned ip address)
8: nfqueue (Linux netfilter queue (NFQUEUE) interface) (Not assigned ip address)
user@KELLGGNLPTP0305:/var/log$ █
```

Рис. 5.4. Вывод команды Packetbeat devices

На рис. 5.4 видим, что существует восемь интерфейсов, из которых можно выбрать наиболее подходящий для мониторинга сетевого трафика или проводить мониторинг со всех интерфейсов.

Чтобы настроить сетевой интерфейс в Packetbeat, необходимо отредактировать файл конфигурации Packetbeat. Ниже приведены шаги настройки.

- Найдите файл конфигурации Packetbeat. По умолчанию он имеет имя `packetbeat.yml` и обычно находится в каталоге `/etc/packetbeat`.
- Откройте файл `packetbeat.yml` в выбранном текстовом редакторе.

Найдите в файле конфигурации раздел интерфейсов. Обычно он выглядит так:

```
interfaces:
  device: any
  type: af_packet
```

- Чтобы настроить конкретный сетевой интерфейс, замените `any` на имя нужного интерфейса. Например, чтобы перехватывать трафик с интерфейса `eth0`, измените конфигурацию следующим образом:

```
interfaces:
  device: eth0
  type: af_packet
```

- Сохраните изменения в файле `packetbeat.yml`.
- Перезапустите Packetbeat, чтобы применить новую конфигурацию.

Команда для перезапуска сервиса может отличаться в зависимости от операционной системы. Например, в Ubuntu используется следующая команда:

```
sudo service packetbeat restart
```

После перезапуска Packetbeat будет перехватывать сетевой трафик с указанного интерфейса в соответствии с настройками, указанными в файле `packetbeat.yml`.

Чтобы отслеживать определенные протоколы в Packetbeat, нужно настроить их в файле конфигурации. Packetbeat поддерживает широкий спектр протоколов, включая DNS, Memcache, ICMP, HTTP и др. Раскомментировав соответствующий протокол и указав при необходимости дополнительные порты, можно настроить параметры мониторинга протоколов.

Вот как может выглядеть раздел протокола в файле конфигурации Packetbeat:

- `protocols:`
- `dns:`
- `ports: [53]`
- `http:`
- `ports: [80, 8080]`

В примере выше протокол DNS настроен на мониторинг трафика на порте 53, а протокол HTTP — на мониторинг трафика на портах 80 и 8080. Чтобы

настроить Packetbeat на захват и анализ протоколов, соответствующих сценарию использования, следует раскомментировать нужные протоколы и изменить номера портов.

Настройка протоколов в файле конфигурации Packetbeat необходима, чтобы Packetbeat осуществлял мониторинг указанных протоколов, предоставляя целевые сведения и анализ сетевого трафика.

На рис. 5.5 показан блок протоколов в файле конфигурации Packetbeat:

```
#===== Transaction protocols =====
packetbeat.protocols:
- type: icmp
  # Enable ICMPv4 and ICMPv6 monitoring. Default: false
  enabled: true
- type: amqp
  # Configure the ports where to listen for AMQP traffic. You can disable
  # the AMQP protocol by commenting out the list of ports.
  ports: [5672]
- type: cassandra
  #Cassandra port for traffic monitoring.
  ports: [9042]
- type: dhcpv4
  # Configure the DHCP for IPv4 ports.
  ports: [67, 68]
- type: dns
  # Configure the ports where to listen for DNS traffic. You can disable
  # the DNS protocol by commenting out the list of ports.
  ports: [53]
- type: http
  # Configure the ports where to listen for HTTP traffic. You can disable
  # the HTTP protocol by commenting out the list of ports.
  ports: [80, 8080, 8000, 5000, 8002]
- type: memcache
  # Configure the ports where to listen for memcache traffic. You can disable
  # the Memcache protocol by commenting out the list of ports.
  ports: [11211]
- type: mysql
  # Configure the ports where to listen for MySQL traffic. You can disable
```

Рис. 5.5. Протоколы Packetbeat

На рис. 5.5 показаны протоколы и порты по умолчанию для них. Протокол можно отключить, добавив комментарий, или включить, раскомментировав. После настройки этих параметров Packetbeat необходимо перезапустить, выполнив следующую команду:

```
sudo service packetbeat restart
```

Эта команда перезапустит Packetbeat, и он начнет получать метрики сервера и отправлять их в кластер Elasticsearch. Документы в индексе можно просмотреть, выполнив такую команду:

```
curl -XGET "http://localhost:9200/.ds-packet-
beat-8.8.1-2023.06.14-000001/_search?
```

Она откроет индекс со списком всех документов. Формат документов следующий:

```
1. {
2.     "_index": ".ds-packetbeat-8.8.1-2023.06.14-000001",
3.     "_id": "nZAku4gBr5DPUDndCHb4",
4.     "_score": 1,
5.     "_source": {
6.         "@timestamp": "2023-06-14T18:21:20.000Z",
7.         "host": {
8.             "containerized": false,
9.             "mac": [
10.                 "18-67-B0-33-01-54",
11.                 "C8-F7-33-F4-B0-6E"
12.             ],
13.             "hostname": "anurag-3570R-370R-470R-450R-510R-4450RV"
14.             "name": "anurag-3570R-370R-470R-450R-510R-4450RV",
15.             "architecture": "x86_64",
16.             "os": {
17.                 "type": "linux",
18.                 "platform": "ubuntu",
19.                 "version": "22.04.2 LTS (Jammy Jellyfish)",
20.                 "family": "debian",
21.                 "name": "Ubuntu",
22.                 "kernel": "5.19.0-43-generic",
23.                 "codename": "jammy"
24.             },
25.             "id": "82cc12b450214f97989ba61ad98e50ce"
26.         },
27.         "network": {
28.             "type": "ipv4",
29.             "transport": "tcp",
30.             "community_id": "1:kgA8l09m6YKGLo0wQnq0jpPU1Tk=",
31.             "bytes": 24551772,
32.             "packets": 3210
33.         },
34.         "destination": {
35.             "port": 9200,
36.             "packets": 1624,
37.             "bytes": 350864,
38.             "ip": "127.0.0.1"
39.         },
40.         "flow": {
41.             "final": false,
```

```
42.          "id": "EAT////AP////CP8AAAF/AAABfwAAASzD8CM".
43.      },
44.      "type": "flow",
45.      "ecs": {
46.          "version": "8.0.0"
47.      },
48.      "agent": {
49.          "type": "packetbeat",
50.          "version": "8.8.1",
51.          "ephemeral_id": "358b3b9a-6a2c-46d0-aab6-22722b882f09",
52.          "id": "832501b5-7b3a-4adc-aa60-d92decc7067c",
53.          "name": "anurag-3570R-370R-470R-450R-510R-4450RV"
54.      },
55.      "source": {
56.          "port": 49964,
57.          "packets": 1586,
58.          "bytes": 24200908,
59.          "ip": "127.0.0.1"
60.      },
61.      "event": {
62.          "duration": 1184160080631,
63.          "dataset": "flow",
64.          "type": "event",
65.          "category": [
66.              "network"
67.          ],
68.          "action": "network_flow",
69.          "type": [
70.              "connection"
71.          ],
72.          "start": "2023-06-14T18:01:30.585Z",
73.          "end": "2023-06-14T18:21:14.745Z"
74.      }
75.  }
76. }
```

Таким образом, Packetbeat можно настроить на чтение данных сетевых пакетов и индексирование их в Elasticsearch.

Получение CSV-данных с помощью Logstash

До сих пор мы обсуждали процесс импорта данных с помощью Beats, изучив сценарии получения логов Apache, данных системных метрик, данных сетевых пакетов и т. д. Теперь рассмотрим возможности Logstash по извлечению CSV-данных. Logstash – универсальный инструмент для извлечения данных из множества источников, включая РСУБД, NoSQL, Kafka, CSV-файлы, Elasticsearch, логи и др. Его роль не ограничивается получением данных и включает их преобразование перед отправкой в место назначения. В состав Logstash входит раздел для получения данных, плагин фильтрации для преобразования данных

и раздел вывода для отправки данных указанным получателям. Такая структура обеспечивает гибкое управление данными. В главе 3 «Elastic Stack: экосистема Elasticsearch» мы подробно познакомились с Logstash, в ней приведено исчерпывающее описание этого инструмента. Сейчас мы сосредоточимся на процессе импорта данных в CSV-файл и их дальнейшей доставки в Elasticsearch. Elasticsearch позволяет применять надежные методы анализа и визуализации данных. Мы будем использовать данные CSV в следующем формате:

```

1. sno, name, age, gender
2. 1, lisa, 32, female
3. 2, sam, 29, male
4. 3, tony, 45, male
5. 4, sonia, 32, female

```

Эти записи находятся в файле `names.csv`, в котором мы отправим их в Elasticsearch. CSV-файл сохранен в директории `Download /home/user/Downloads/`, но его расположение может быть любым.

Примечание. Помните, что файл должен быть доступен через Logstash.

Теперь нужно создать файл конфигурации Logstash для чтения этих CSV-данных из файла `names.csv`. Создадим файл конфигурации `pull_names.conf` Logstash и добавим в него следующий код:

```

1. input {
2.   file {
3.     path => "/home/user/Downloads/names.csv"
4.     start_position => beginning
5.   }
6. }
7. filter {
8.   csv {
9.     autodetect_column_names => true
10.  }
11. }
12. output {
13.   stdout
14.   {
15.     codec => rubydebug
16.   }
17. elasticsearch {
18.   action => "index"
19.   hosts => ["127.0.0.1:9200"]
20.   index => "names"
21.   user => elastic
22.   password => your_password
23. }
24. }

```

В предыдущем фрагменте кода показан код конфигурации Logstash, содержащий три блока: ввод, фильтр и вывод. В блоке ввода используется плагин `file` для чтения данных из файла. Здесь мы указываем путь к файлу CSV и задаем начало файла в `start_position`, чтобы Logstash начинал каждое выполнение конфигурации для получения данных CSV с начала файла.

Второй блок — фильтр, в котором используется плагин CSV. Здесь мы устанавливаем для `autodetect_column_names` значение `true`, чтобы Logstash автоматически определял имена столбцов из CSV-файла. Также можно указать имена полей вручную, как показано ниже:

```

1. filter {
2.     csv {
3.         columns => [
4.             "id",
5.             "Name",
6.             "Age",
7.             "Gender"
8.         ]
9.         separator => ","
10.    }
11. }
```

В коде выше имена полей указаны вручную. Таким образом, можно изменить имя метки, но при автоопределении Logstash будет использовать имя, указанное в CSV-файле.

Третий блок — это блок вывода, где используется плагин `stdout` для отправки данных на стандартный вывод. Для форматирования стандартного вывода используется кодек `rubydebug`. Также используется плагин Elasticsearch для отправки CSV-данных в Elasticsearch. В плагине Elasticsearch необходимо задать действие, хосты, имя индекса, имя пользователя и пароль Elasticsearch.

Итак, мы рассмотрели создание файла конфигурации Logstash для получения CSV-данных, отправки их в Elasticsearch и вывода записей на терминале.

Теперь нужно запустить этот файл с домашней страницы Logstash, используя следующую команду:

```
bin/logstash -f /etc/logstash/conf.d/pull_names.conf
```

Команда извлечет данные из CSV-файла и отправит их на стандартный вывод и в Elasticsearch. После ее выполнения увидим следующий вывод в терминале:

```

1. {
2.     "name" => " lisa",
3.     "@version" => "1",
```

```

4.      "@timestamp" => 2023-06-15T12:08:33.436Z,
5.      "gender" => "female",
6.      "id" => "1",
7.      "host" => "KELGGNLPTP0305",
8.      "message" => "1, lisa,32, female",
9.      "path" => "/home/user/Downloads/names.csv",
10.     "age" => "32"
11. }
```

Эти данные в формате JSON выводятся на терминал с помощью плагина `stdout`.

Теперь выполним следующую команду и сформируем список индексов Elasticsearch, чтобы проверить, создан ли в Logstash индекс имен:

```
curl -XGET "http://localhost:9200/_cat/indices?v"
```

Команда выведет список всех индексов, и мы сможем найти в нем индекс имен `names`. Далее можно просмотреть документы индекса, выполнив следующее:

```
curl -XGET "http://localhost:9200/names/_search?"
```

Эта команда откроет индекс со списком всех документов. Вот формат документа:

```

1. {
2.     "_index" : "names",
3.     "_type" : "_doc",
4.     "_id" : "vqXLOnEBGEvdm4dfC8T7",
5.     "_score" : 1.0,
6.     "_source" :
7.     {
8.         "Name" : "sonia",
9.         "@version" : "1",
10.        "@timestamp" : "2023-06-15T12:08:33.443Z",
11.        "Gender" => "female",
12.        "id" : "4",
13.        "host" : "KELGGNLPTP0305",
14.        "message" : "4, sonia, 32, female",
15.        "path" : "/home/user/Downloads/names.csv",
16.        "age" : "32"
17.    }
18. }
```

Приведенная выше запись в формате JSON — пример документа Elasticsearch из индекса `names`, созданного в Logstash из файла `names.csv`. Таким же образом с помощью Logstash можно получить данные из любого файла, а также записи из любой базы данных RDBSM или NoSQL либо любого другого источника.

ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели, почему данные так важны для решения бизнес-задач. Затем мы разобрались, что такое поглощение, доставка, хранение и визуализация данных. Мы также рассмотрели различные способы импорта данных в Elasticsearch и на практических примерах изучили принципы работы инструментов Beats, таких как Filebeat, Metricbeat и Packetbeat. Кроме того, мы узнали, как настроить Logstash для отправки данных в формате CSV в Elasticsearch.

В следующей главе вы узнаете, как создать индекс и маппинг. Затем мы перейдем к управлению индексами, выполнению операций на уровне индексов, управлению шаблонами индексов, API-интерфейсам индексов и управлению жизненным циклом индексов.

ВОПРОСЫ

1. Объясните важность данных для бизнеса.
2. Что такое доставка данных?
3. Что такое поглощение данных?
4. Импортируйте логи веб-сервера с помощью Filebeat.
5. Импортируйте метрики сервера с помощью Metricbeat.
6. Извлеките данные из CSV-файла с помощью Logstash.

ГЛАВА 6

Управление индексами: создание, обновление и удаление индексов Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели фундаментальные концепции поглощения данных, их доставки, хранения и визуализации. Мы изучили различные методы импорта данных в Elasticsearch, рассмотрели практические реализации основных инструментов Beats, таких как Filebeat, Metricbeat и Packetbeat. Кроме того, мы разобрались, как происходит поглощение данных с помощью Logstash. В этой главе мы остановимся на управлении индексами в Elasticsearch. Сначала изучим создание индексов и особенности настройки маппинга. Затем перейдем к управлению индексами Elasticsearch и познакомимся с основными методами выполнения операций на уровне индексов. Кроме того, мы исследуем управление шаблонами индексов и разнообразные API-интерфейсы уровня индексов, позволяющие выполнять множество операций с индексами. И наконец, в завершение освоим тонкости управления жизненным циклом индекса.

СТРУКТУРА

В этой главе:

- Введение в создание и маппинг индексов Elasticsearch
- Управление индексами в Elasticsearch
- Выполнение операций над индексами Elasticsearch

- Индексные API Elasticsearch
- Удаление индексов
- Управление шаблонами индексов Elasticsearch
- Управление жизненным циклом индекса в Elasticsearch

ЦЕЛИ

По завершении этой главы вы приобретете навыки и знания, необходимые для эффективного управления индексами Elasticsearch. Вы научитесь создавать индексы и выполнять с ними различные операции. Вы также научитесь настраивать жизненный цикл индексов — важнейший элемент оптимизации управления и хранения данных в Elasticsearch.

ВВЕДЕНИЕ В СОЗДАНИЕ И МАППИНГ ИНДЕКСОВ ELASTICSEARCH

Индексы в Elasticsearch служат для упорядочивания и хранения однотипных записей по аналогии с таблицами в реляционной базе данных. Elasticsearch позволяет создавать несколько индексов, каждый из которых может содержать несколько документов. Создать индекс в Elasticsearch можно с помощью *API создания индекса*, реализуемого через запрос PUT. Методы создания индекса Elasticsearch могут быть разными, например, создание пустого индекса без связанных с ним документов или индексирование документа, который впоследствии запускает процесс создания индекса. Кроме того, индексы можно создавать путем передачи данных из различных источников, например, с помощью Beats или Logstash. В предыдущей главе мы рассмотрели, как автоматизировать создание индексов Elasticsearch с помощью Beats и Logstash.

Создание индекса без документов

Теперь посмотрим, как создать индекс Elasticsearch, не добавляя в него документы. Для этого необходимо выполнить такую команду:

```
curl -XPUT "http://localhost:9200/testindex"
```

Эта команда создает тестовый индекс `testindex` и выводит следующий результат:

```
1. {
2.   "acknowledged" : true,
3.   "shards_acknowledged" : true,
```

```

4.   "index" : "testindex"
5. }
```

Этот ответ в формате JSON содержит имя индекса, параметр `acknowledged`, имеющий значение `true`, и параметр `shard_acknowledged` также со значением `true`. Если значение `acknowledged` оказывается `true`, индекс создан успешно. Подробную информацию об индексе можно получить, выполнив следующую команду:

```
curl -XGET "http://localhost:9200/testindex"
```

Ответ будет таким:

```

1. {
2.   "testindex": {
3.     "aliases": {},
4.     "mappings": {},
5.     "settings": {
6.       "index": {
7.         "routing": {
8.           "allocation": {
9.             "include": {
10.               "_tier_preference": "data_content".
11.             }
12.           }
13.         },
14.         "number_of_shards": "1",
15.         "provided_name": "testindex",
16.         "creation_date": "1687192322272",
17.         "number_of_replicas": "1",
18.         "uuid": "kXQziCVNQE-mCienmG6tqw",
19.         "version": {
20.           "created": "8070199"
21.         }
22.       }
23.     }
24.   }
25. }
```

Этот ответ содержит имя индекса, маппинг, который является пустым, поскольку при создании индекса мы не указывали подробности, количество шардов, дату создания, UUID, версию, количество реплик и т. д.

Если попытаться создать индекс, который уже существует, Elasticsearch выдаст ошибку `400`. Вот ответ, который Elasticsearch вернет при попытке создать тот же индекс снова, используя предыдущую команду `create index`:

```

1. {
2.   "error": {
3.     "root_cause": [
```

```
4.      {
5.          "type": "resource_already_exists_exception",
6.          "reason": "index [testindex/kXQziCVNQE-mCienmG6tqw] already exists",
7.          "index_uuid": "kXQziCVNQE-mCienmG6tqw",
8.          "index": "testindex"
9.      }
10. ],
11. "type": "resource_already_exists_exception",
12. "reason": "index [testindex/kXQziCVNQE-mCienmG6tqw] already exists",
13. "index_uuid": "kXQziCVNQE-mCienmG6tqw",
14. "index": "testindex"
15. },
16. "status": 400
17. }
```

В полученном ответе видим, что первопричина ошибки представлена в виде JSON-документа. В документе указано, что тип ошибки — `"resource_already_exists_exception"`, а причина — `"index [testindex/kXQziCVNQE-mCienmG6tqw] already exists"`. При попытке создать индекс, который уже был создан, Elasticsearch выдаст ошибку 400 вместе с этим сообщением об ошибке. Такой механизм позволяет обнаружить и надлежащим образом обработать создание дубликатов.

Создание индекса, содержащего документы

Elasticsearch позволяет автоматически создать индекс при публикации первого документа без предварительного явного создания индекса. Рассмотрим, как создать индекс сразу с документами, в отличие от предыдущего примера, без документов. Ниже показана команда, используемая для создания индекса с документом:

```
1. curl -XPOST "http://localhost:9200/testindex/_doc/1" -H 'Content-Type:
   application/json' -d '{
2.     "name": "Sophia",
3.     "gender": "female",
4.     "city": "Singapore"
5. }'
```

В качестве альтернативы можно выполнить такую команду в формате JSON непосредственно из Kibana:

```
1. POST testindex/_doc/1
2. {
3.     "name": "Sophia",
4.     "gender": "female",
5.     "city": "Singapore"
6. }
```

Эта команда создает индекс и вставляет в него документ с идентификатором 1. Чтобы извлечь документ, выполним команду:

```
curl -XGET "http://localhost:9200/testindex/_doc/1/"
```

Она извлекает документ с указанным идентификатором. Выполнив команду, получим следующий ответ:

```
1. {
2.   "_index": "testindex",
3.   "_type": "_doc",
4.   "_id": "1",
5.   "_version": 2,
6.   "_seq_no": 2,
7.   "_primary_term": 1,
8.   "found": true,
9.   "_source": {
10.     "name": "Sophia",
11.     "gender": "female",
12.     "city": "Singapore"
13.   }
14. }
```

Стоит отметить, что при прямом индексировании документов без явного определения маппинга существует потенциальная проблема. Elasticsearch пытается предположить тип данных полей индексируемого документа, но делает это не всегда точно. Поэтому, чтобы обеспечить точное индексирование и запрос данных, рекомендуется сопоставлять поля с соответствующими им типами данных.

Получение маппинга индекса

В Elasticsearch под маппингом понимается процесс определения схемы или структуры документов, которые будут храниться в индексе. Она определяет, как будет индексироваться и храниться каждое поле в документе, а также его тип данных, настройки анализа и другие свойства.

В Elasticsearch маппинг очень важен для обработки данных и выполнения эффективных операций поиска, анализа и агрегирования. Определяя маппинг, мы устанавливаем порядок токенизации, нормализации и индексирования полей, что позволяет получать точные и релевантные результаты поиска.

Чтобы получить информацию о маппинге индекса, можно воспользоваться такой командой:

```
curl -XGET "http://localhost:9200/testindex/_mapping"
```

Команда выводит сведения о маппинге индекса "testindex". Результат ее выполнения следующий:

```
1. {
2.   "testindex": {
3.     "mappings": {
4.       "properties": {
5.         "city": {
6.           "type": "text",
7.           "fields": {
8.             "keyword": {
9.               "type": "keyword",
10.              "ignore_above": 256
11.            }
12.          }
13.        },
14.        "gender": {
15.          "type": "text",
16.          "fields": {
17.            "keyword": {
18.              "type": "keyword",
19.              "ignore_above": 256
20.            }
21.          }
22.        },
23.        "name": {
24.          "type": "text",
25.          "fields": {
26.            "keyword": {
27.              "type": "keyword",
28.              "ignore_above": 256
29.            }
30.          }
31.        }
32.      }
33.    }
34. }
```

Приведенный выше вывод JSON представляет собой маппинг индекса "testindex". Он содержит имена полей, соответствующие им типы данных, а также информацию о том, относятся ли поля к типу "keyword".

В параметрах маппинга содержится информация о структуре индексированных данных, позволяющая определить тип данных полей. Эта информация очень важна для точного запроса и анализа данных, хранящихся в индексе.

Создание маппинга

Чтобы создать маппинг для индекса в Elasticsearch, выполните следующие действия:

1. Создайте индекс, отправив запрос PUT с нужным именем индекса:

```
PUT newindex
```

Эта команда создает индекс "newindex".

2. Задайте маппинг полей в индексе, отправив запрос PUT на конечную точку _mappings:

```

1. PUT newindex/_mappings
2. {
3.   "properties": {
4.     "firstname": {
5.       "type": "keyword"
6.     },
7.     "lastname": {
8.       "type": "keyword"
9.     },
10.    "account_number": {
11.      "type": "integer"
12.    },
13.    "balance": {
14.      "type": "integer"
15.    },
16.    "age": {
17.      "type": "integer"
18.    },
19.    "gender": {
20.      "type": "keyword"
21.    }
22.  }
23. }
```

В этом примере мы задали маппинг для индекса "newindex" с набором полей. Поля `firstname`, `lastname` и `gender` отображаются как ключевые слова, что позволяет искать их как точные термины. Поля `account_number`, `balance` и `age` отображаются как целые числа.

3. После создания маппинга его можно получить, отправив запрос GET на конечную точку `_mapping`:

```
GET newindex/_mapping
```

Эта команда извлекает маппинг индекса "newindex" и возвращает ответ в формате JSON с указанием типа данных для каждого поля.

Явный маппинг гарантирует, что Elasticsearch верно интерпретирует и обрабатывает данные в индексе. Такой подход обеспечивает точный поиск и анализ данных.

УПРАВЛЕНИЕ ИНДЕКСАМИ В ELASTICSEARCH

Удобный доступ к управлению индексами осуществляется через пользовательский интерфейс Kibana, предоставляющий полный обзор настроек индекса,

маппингов, статистики и пр. Чтобы получить доступ к управлению индексами в Kibana, выполните следующие действия:

1. Откройте Kibana и перейдите по ссылке **Management** (Управление) в левом боковом меню. Нажав на нее, вы попадете на страницу управления **Management**.
2. На странице **Management** найдите опцию **Index Management** (Управление индексами) и щелкните по ней. Откроется страница управления индексами.
3. На странице **Index Management** собрана информация и сведения о функциональных возможностях имеющихся индексов, в том числе возможность просматривать и изменять настройки индекса, определять маппинг, отслеживать статистику индекса, выполнять операции на уровне индекса и решать задачи, связанные с управлением индексом.

Интуитивно понятный интерфейс Kibana позволяет эффективно управлять индексами и анализировать их, обеспечивая оптимальную конфигурацию и производительность в среде Elasticsearch (рис. 6.1).

The screenshot shows the 'Index Management' page in Kibana. At the top, there are tabs for 'Indices' and 'Index Templates'. Below that is a search bar and filter options for 'Lifecycle status' and 'Reload Indices'. The main area displays a table of indices with the following columns: Name, Health, Status, Primaries, Replicas, Docs count, and Storage size. The indices listed are:

Name	Health	Status	Primaries	Replicas	Docs count	Storage size
popular_baby	yellow	open	1	1	11346	1.7mb
my_index_term	yellow	open	1	1	1	3.3kb
twitter	yellow	open	1	1	2	4.8kb
filebeat-7.6.1	yellow	open	1	1	7	97.6kb
mondata	yellow	open	1	1	4	25.1kb

Рис. 6.1. Управление индексами

На рис. 6.1 показана страница управления индексами в Kibana, содержащая обзор индексов, включающий такие важные данные, как информация о состоянии, количество первичных шардов и реплик, количество документов, размер хранилища и т. д. На странице указываются операции, которые можно выполнять с индексами, включая просмотр настроек индекса, разметок, статистики, а также редактирование настроек индекса, закрытие, сброс на диск и удаление индекса.

С помощью управления индексами можно выполнять различные операции на уровне индексов, такие как обновление, принудительное объединение

сегментов, замораживание индексов, очистка кэша, сброс на диск и многие другие. Также можно выполнять операции над несколькими индексами одновременно. Отметим, что индексы в списке на странице управления индексами в Kibana сопровождаются значками, указывающими, является ли индекс замороженным, свернутым или индексом-последователем. Эти значки можно использовать для фильтрации списка индексов, а также для улучшения управления и организации индексов в системе.

ВЫПОЛНЕНИЕ ОПЕРАЦИЙ НАД ИНДЕКСАМИ ELASTICSEARCH

На уровне индекса можно выполнить ряд операций, используя опцию **Manage index** на странице Index Management (рис. 6.2).

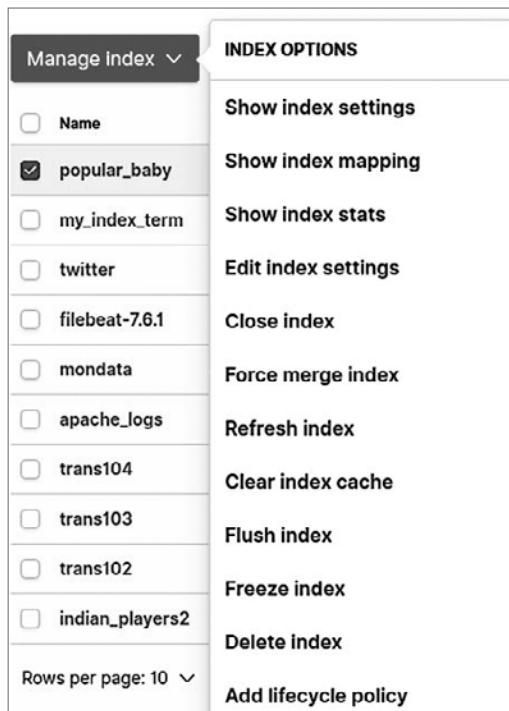


Рис. 6.2. Всплывающее окно Manage index

На рис. 6.2 показан экран с выпадающим списком **Manage index**. Используя опцию **Manage index** на странице управления индексами в Kibana, можно

выполнять операции как с отдельным индексом, так и с несколькими. Ниже перечислены эти операции.

Закрытие индекса

Операция `Close index` (Закрыть индекс) в Kibana позволяет заблокировать индекс, сделав его недоступным для чтения и записи. Эту операцию можно выполнить для одного или нескольких индексов. Когда индекс закрыт, он занимает только дисковое пространство и не использует никаких других ресурсов. При необходимости вернуть доступ к индексу его можно открыть снова, запустив стандартный процесс восстановления.

Удаление индекса

С помощью операции `Delete index` (Удалить индекс) в Kibana можно удалить один или несколько индексов. Эта операция удаляет не только индексы, но и все содержащиеся в них документы. Для выполнения этой операции Kibana предоставляет REST API, которые будут рассмотрены в следующем разделе. Удаление индекса необратимо, поэтому следует соблюдать осторожность и перед удалением убедиться, что данные больше не нужны.

Замораживание индекса

Операция `Freeze index` (Заморозить индекс) в Kibana делает индекс доступным только для чтения. При замораживании индекса его шарды перемещаются на диск, что приводит к уменьшению занимаемой памяти. Хотя по замороженным индексам по-прежнему можно выполнять операции поиска, может возникнуть небольшая задержка, пока Elasticsearch извлекает результаты с диска. Операции записи для замороженных индексов блокируются, и индексы сохраняют состояние «только для чтения». Замораживание индекса может быть полезно для оптимизации использования ресурсов, особенно в случае с индексами, в которых хранятся преимущественно архивные или исторические данные.

Обновление индекса

Операция `Refresh index` (Обновить индекс) явно запускает обновление одного или нескольких индексов в Elasticsearch. По умолчанию Elasticsearch автоматически обновляет индексы каждую секунду. Однако операция `Refresh index` позволяет запускать ручное обновление в любое время. Важно отметить, что автоматическое обновление выполняется только в том случае, если к индексу был направлен один или больше поисковых запросов за последние 30 секунд, что определяется параметром `index.refresh_interval`. При необходимости этот интервал можно изменить.

Хотя операцию обновления можно инициировать вручную, она требует больших ресурсов и ее не рекомендуется делать регулярной. Лучше всего использовать периодическое автоматическое обновление Elasticsearch. Операция обновления делает недавно проиндексированные документы доступными для поиска, гарантируя, что изменения в индексе будут видны в результатах поиска. Явно обновляя индекс, мы сразу делаем доступными для поиска все последние обновления или дополнения.

Принудительное слияние индекса

Операция `Force merge index` (Принудительное слияние) объединяет меньшие файлы шардов в индексе, уменьшая количество сегментов и оптимизируя производительность. Эта операция особенно полезна, если индекс доступен только для чтения. При принудительном слиянии удаленные документы также очищаются, освобождая место на диске. Стоит отметить, что принудительное слияние может быть ресурсоемкой операцией, поэтому важно учитывать размер и характер индекса перед ее выполнением.

Очистка кэша индекса

Операция `Clear index cache` (Очистить кэш индекса) очищает все кэши, связанные с индексом. Для повышения производительности поиска Elasticsearch использует разные кэши, такие как кэш данных полей, кэш фильтров и кэш запросов. Очистка кэша позволяет освободить память и гарантировать, что в ходе проведенного после нее поиска из индекса будут получены самые актуальные данные.

Сброс индекса на диск

Операция `Flush index` (Сбросить на диск) синхронизирует кэш файловой системы индекса с диском. Эта операция обеспечивает постоянное хранение всех данных лога транзакций в индексе Lucene. Сброс индекса на диск освобождает память и помогает поддерживать целостность данных. Эта операция может быть ресурсоемкой, и ее частота должна определяться исходя из конкретных требований развертывания Elasticsearch.

Добавление политики жизненного цикла

Операция `Add lifecycle policy` (Добавить политику жизненного цикла) управления индексами Kibana настраивает политики **управления жизненным циклом индекса** (Index Lifecycle Management, ILM). ILM предоставляет механизм для определения действий, выполняемых над индексом по мере увеличения его

возраста. Эта функция особенно полезна для управления данными временных рядов, такими как логи, собранные Beats.

Политика ILM устанавливает условия и действия, определяющие, когда и как управлять индексом. Например, можно настроить политику на автоматический переход к новому индексу, когда существующий индекс достигает определенного размера или возраста. Реализуя перенос индекса, можно поддерживать меньшие, более управляемые индексы, оптимизированные для производительности поиска.

Основные преимущества использования политик ILM:

- **Управление размером индекса.** Определяя условия на основе размера индекса, можно избежать чрезмерного разрастания индексов, влияющего на производительность поиска и потребление ресурсов. Перенос на новые индексы позволяет лучше организовать данные и упростить управление ими.
- **Действия на основе возраста.** Политики ILM также могут учитывать возраст индекса при определении действий. Например, можно указать, что по истечении определенного срока индекс должен быть заморожен, удален или перемещен на более медленные уровни хранения. Это позволяет эффективно управлять жизненным циклом данных на основе требований к хранению.
- **Автоматизация и масштабируемость.** Политики ILM автоматизируют управление индексами, снижая необходимость ручного вмешательства. Это особенно полезно при работе с крупными развертываниями с большим количеством индексов. Политики могут применяться к нескольким индексам одновременно, что обеспечивает эффективное и последовательное управление в кластере Elasticsearch.

Настроив политики жизненного цикла в Kibana, можно определить желаемое поведение индексов по мере увеличения их возраста. Это обеспечивает эффективное хранение данных, а также оптимизацию поиска и управления данными временных рядов.

API ELASTICSEARCH ДЛЯ РАБОТЫ С ИНДЕКСАМИ

Elasticsearch предлагает полный набор REST-интерфейсов, позволяющих выполнять различные операции над индексами. Как уже говорилось, эти операции содержатся в меню **Index Management** в Kibana. Однако, используя API индексов, пользователи могут получить доступ к более широкому спектру возможностей, превосходящих доступные в пользовательском интерфейсе управления. Эти API позволяют управлять отдельными индексами, включая работу с маппингом, настройку шаблонов индексов, определение псевдонимов и настройку параметров индексов, а также другие функции.

Используя API для работы с индексами, пользователи могут программно управлять задачами, связанными с индексом, и настраивать их. Это позволяет разработчикам и администраторам автоматизировать процессы, легко интегрировать Elasticsearch с другими системами и выполнять расширенные операции в соответствии с конкретными требованиями.

В предыдущем разделе мы обсудили некоторые операции на уровне индексов, которые можно выполнять из меню **Index Management** в Kibana. Все эти операции можно выполнять и с помощью API, причем API предоставляют больше возможностей, чем пользовательский интерфейс управления. API для работы с индексами позволяют управлять отдельными индексами, их маппингом, шаблоном индекса, псевдонимами, настройками индекса и т. д. В этой главе мы не будем рассматривать все API, обсудим лишь наиболее важные из них. Начнем с API управления индексами.

Управление индексами

Elasticsearch предоставляет ряд API, которые облегчают управление и манипулирование индексами. Эти API позволяют выполнять такие действия, как создание, удаление индекса, получение информации об индексе, закрытие и повторное открытие индекса, клонирование и замораживание индекса.

Создание индекса

При создании нового индекса с помощью API можно применять различные опции и конфигурации. Помимо указания имени индекса строчными буквами, API позволяет выполнять дополнительные операции, такие как изменение настроек индекса, определение маппинга индекса и создание псевдонимов индекса.

Для создания индекса используется команда

```
PUT /<index_name>
```

При присвоении имени индексу важно придерживаться следующих критериев:

- имя индекса должно быть написано в нижнем регистре;
- имя не может содержать такие символы, как #, , , /, *, ?, ", <, >, | и (символ пробела);
- не допускается начинать имя индекса с символов _, – или +;
- длина имени индекса не должна превышать 255 байт.

API создания индекса также поддерживает параметры запроса, которые можно использовать для установки опций при создании индекса. К таким параметрам

относятся `include_type_name`, `wait_for_active_shards`, `timeout`, `master_timeout` и др.

Вот что означает каждый из этих параметров:

- `include_type_name` — определяет, включать ли имя типа в ответ API. По умолчанию Elasticsearch версий 7.0 и выше не требует указания имени типа, поэтому рекомендуется установить значение `include_type_name` в `false`.
- `wait_for_active_shards` — контролирует количество активных шардов, необходимое для возврата операции создания индекса. Он задает, какое количество копий шардов должно быть активно, чтобы индекс считался успешно созданным.
- `timeout` — задает время ожидания завершения операции создания индекса. Если установленное время истечет до завершения операции, Elasticsearch вернет ответ, указывающий на тайм-аут.
- `master_timeout` — определяет максимальное время ожидания ответа от главного узла, прежде чем считать его тайм-аутом. Это особенно важно, когда запрос включает изменения состояния кластера.

Это лишь несколько примеров параметров запроса, доступных для API создания индекса. Elasticsearch предоставляет опции для тонкой настройки процесса создания индекса и учета специфических требований или соображений производительности.

Используя эти параметры, в процессе создания индекса можно применять определенные конфигурации. Кроме того, тело запроса может выполнять дополнительные операции при создании индекса. Тело запроса поддерживает такие параметры, как псевдонимы, маппинг и настройки. С их помощью можно создать псевдонимы для индекса, а также определить маппинг или изменить настройки индекса.

Используя API создания индексов и его параметры, Elasticsearch обеспечивает гибкость и контроль над созданием и конфигурацией индексов, позволяя пользователям настраивать их в соответствии с конкретными потребностями.

Удаление индекса

API удаления индексов позволяет удалить индекс из Elasticsearch с помощью указанной конечной точки:

```
DELETE /<index_name>
```

Удалить индекс можно, отправив запрос `DELETE` с именем индекса. Чтобы удалить все индексы, можно использовать в запросе подстановочное выражение `_all` или `*`.

API удаления индексов поддерживает несколько параметров запроса для настройки операции удаления:

- `allow_no_indices` — определяет, должен ли API возвращать ошибку, если указанный индекс не существует. По умолчанию имеет значение `true`, это означает, что API не будет выдавать ошибку, если индекс не найден.
- `expand_wildcard` — управляет поведением подстановочного выражения. Если этот параметр имеет значение `open` или `closed`, он расширяет подстановочное выражение для сопоставления с открытыми или закрытыми индексами соответственно.
- `ignore_unavailable` — указывает, следует ли игнорировать недоступные индексы. Если установлено значение `true`, API будет игнорировать все недоступные индексы, поскольку они закрыты и не содержат шардов.
- `timeout` — задает время ожидания завершения операции удаления. Если операция выполняется дольше указанного времени, Elasticsearch выдает сообщение об ошибке с указанием тайм-аута.
- `master_timeout` — определяет максимальное время ожидания ответа от главного узла, прежде чем считать его тайм-аутом. Это особенно важно, когда запрос включает изменения состояния кластера.

Указанные параметры запроса обеспечивают гибкость и контроль удаления индекса. За полным списком доступных параметров и подробностями их использования обращайтесь к документации Elasticsearch.

Получение индекса

API получения индекса позволяет получить информацию об одном или нескольких индексах в Elasticsearch. Выражение API для получения информации об индексе выглядит так:

```
GET /<index_name>
```

Чтобы получить информацию о конкретном индексе, необходимо отправить запрос GET с именем индекса. Чтобы получить информацию обо всех индексах, вместо конкретного имени индекса используется параметр пути `_all`. Кроме того, чтобы получить сведения о конкретных индексах, можно указать список индексов, разделенных запятыми.

API получения индексов поддерживает набор параметров запроса для настройки поведения API:

- `allow_no_indices` — определяет, должен ли API возвращать ошибку, если указанный индекс не существует. Если установить значение `true`, API не будет выдавать ошибку и вернет пустой ответ, если индекс не будет найден.

- `expand_wildcards` — управляет поведением подстановочных выражений для имен индексов. Он позволяет указать, какой тип индексов должен быть расширен. Возможные значения: открытый, закрытый, скрытый, никакой или их комбинация.
- `flat_settings` — указывает, возвращать ли настройки индекса в плоском формате. Если установлено значение `true`, настройки будут возвращены в виде плоской структуры.
- `include_defaults` — определяет, включать ли в ответ настройки индекса по умолчанию. Если установлено значение `true`, в ответ будут включены настройки по умолчанию в дополнение к пользовательским настройкам.
- `include_type_name` — указывает, включать ли тип имени в ответ. Этот параметр устарел и будет удален в следующих версиях Elasticsearch.
- `ignore_unavailable` — управляет тем, как API обрабатывает недоступные индексы. Если установлено значение `true`, API будет игнорировать недоступные индексы, поскольку они закрыты или в них отсутствуют шарды.
- `local` — определяет, должен ли API получать информацию только с локального узла. Если установлено значение `true`, запрос будет выполняться локально на узле-получателе.
- `master_timeout` — определяет максимальное время ожидания ответа от главного узла, прежде чем считать его тайм-аутом. Это особенно важно, когда запрос включает изменения состояния кластера.

Указанные параметры запроса обеспечивают гибкость настройки поведения API получения индекса. За полным списком доступных параметров и подробностями их использования обращайтесь к документации Elasticsearch.

Закрытие индекса

API закрытия индекса закрывает открытый индекс в Elasticsearch. Чтобы закрыть индекс, необходимо отправить запрос POST с именем индекса и параметром `_close`. Выражение API для закрытия индекса выглядит так:

```
POST /<index_name>/_close
```

Эта команда служит для закрытия индекса, и результат ее выполнения следующий:

```
1. {
2.     "acknowledged" : true,
3.     "shards_acknowledged" : true,
4.     "indices" : {
5.         "<index_name>" : {
```

```

6.           "closed" : true
7.       }
8.   }
9. }
```

Когда индекс закрыт, он становится недоступен для чтения и записи. Он фактически блокируется от любых действий по индексированию или поиску. Чтобы закрыть все индексы, в качестве имени индекса в вызове API используется `_all`.

API закрытия индекса также поддерживает различные параметры запроса, которые можно использовать для изменения поведения операции:

- `allow_no_indices` — определяет, должен ли API возвращать ошибку, если указанный индекс не существует. Если установить значение `true`, API не будет выдавать ошибку и вернет пустой ответ, если индекс не будет найден.
- `expand_wildcards` — управляет поведением подстановочных выражений для имен индексов. Он позволяет указать, какие типы индексов должны быть расширены. Возможные значения: открытый, закрытый, скрытый, никакой или их комбинация.
- `ignore_unavailable` — управляет тем, как API обрабатывает недоступные индексы. Если установлено значение `true`, API будет игнорировать недоступные индексы, поскольку они закрыты или в них отсутствуют шарды.
- `wait_for_active_shards` — определяет, какое количество активных шардов необходимо для выполнения операции. Прежде чем закрыть индекс, API будет ожидать, пока указанное количество шардов не станет доступным.
- `timeout` — определяет максимальное время ожидания завершения операции, прежде чем считать ее тайм-аутом.
- `master_timeout` — определяет максимальное время ожидания ответа от главного узла, прежде чем считать его тайм-аутом. Это особенно важно, когда запрос включает изменения состояния кластера.

Если потребуется выполнить какие-либо операции, такие как поиск, индексирование или обновление, с закрытым индексом, необходимо будет сначала открыть его с помощью API открытия индекса.

Открытие индекса

API открытия индекса в Elasticsearch открывает закрытый индекс для чтения и записи. Чтобы открыть индекс, необходимо отправить запрос POST с именем индекса и параметром `_open`. Выражение API для открытия индекса выглядит так:

```
POST /<index_name>/_open
```

Чтобы открыть закрытый индекс, необходимо указать имя индекса в вызове API. Чтобы открыть все закрытые индексы, в качестве имени индекса используется `_all`.

API открытия индекса также поддерживает различные параметры запроса, которые можно использовать для изменения поведения операции:

- `allow_no_indices` — определяет, должен ли API возвращать ошибку, если указанный индекс не существует. Если установить значение `true`, API не будет выдавать ошибку и вернет пустой ответ, если индекс не будет найден.
- `expand_wildcards` — управляет поведением подстановочных выражений для имен индексов. Он позволяет указать, какие типы индексов должны быть расширены. Возможные значения: открытый, закрытый, скрытый, никакой или их комбинация.
- `ignore_unavailable` — управляет тем, как API обрабатывает недоступные индексы. Если установлено значение `true`, API будет игнорировать недоступные индексы, поскольку они закрыты или в них отсутствуют шарды.
- `wait_for_active_shards` — определяет, какое количество активных шардов необходимо для выполнения операции. Прежде чем открыть индекс, API будет ожидать, пока указанное количество шардов не будет доступно.
- `timeout` — определяет максимальное время ожидания завершения операции, прежде чем считать ее тайм-аутом.
- `master_timeout` — определяет максимальное время ожидания ответа от главного узла, прежде чем считать его тайм-аутом. Это особенно важно, когда запрос включает изменения состояния кластера.

После открытия индекс становится доступным для чтения и записи. В индексе можно выполнять такие действия, как индексирование документов, поиск или обновление данных.

API проверки наличия индекса

API проверки наличия индекса в Elasticsearch позволяет проверить доступность индекса в кластере. Определить, существует ли индекс, можно, отправив запрос `HEAD` с именем индекса. Выражение API для проверки существования индекса выглядит так:

```
HEAD /<index_name>
```

После выполнения этой команды API вернет код состояния и сообщение. Код ответа `200` означает, что индекс или псевдонимы существуют в кластере, а код ответа `404` — что указанный индекс или псевдонимы недоступны.

Например, чтобы проверить существование индекса с именем `newindex`, необходимо выполнить следующую команду:

```
HEAD /newindex
```

Ответ может быть таким:

```
200 – OK
```

Код ответа `200` подтверждает, что индекс доступен в кластере Elasticsearch. Чтобы проверить статус нескольких индексов, в параметре `path` необходимо указать список их имен, разделенных запятыми. API для проверки наличия индекса также поддерживает различные параметры запроса, изменяющие поведение операции:

- `allow_no_indices` — определяет, должен ли API возвращать ошибку, если ни один из указанных индексов не существует. Если установить значение `true`, API не будет выдавать ошибку и вернет пустой ответ, если ни один из индексов не будет найден.
- `expand_wildcards` — управляет поведением подстановочных выражений для имен индексов. В нем можно указать, какие типы индексов должны быть расширены. Возможные значения: открытый, закрытый, скрытый, ни один или их комбинация.
- `ignore_unavailable` — управляет тем, как API обрабатывает недоступные индексы. Если установить значение `true`, API будет игнорировать недоступные индексы, поскольку они закрыты или в них отсутствуют шарды.
- `flat_settings` — определяет, возвращать ли настройки в плоском формате. Если установлено значение `true`, настройки будут возвращены в виде упрощенной плоской структуры.
- `include_defaults` — определяет, возвращать ли в ответ настройки по умолчанию. Если установлено значение `true`, настройки по умолчанию будут включены.
- `local` — указывает, следует ли выполнять запрос локально на узле, который его получил, или передавать другим узлам кластера.

Эти параметры запроса позволяют настроить поведение API проверки наличия индекса в соответствии с заданными потребностями.

Сжатие индекса

Чтобы уменьшить существующий индекс в Elasticsearch, можно использовать API сжатия индекса. Он позволяет уменьшить количество первичных шардов

в индексе, что приводит к уменьшению его размера. Для уменьшения индекса используется следующее выражение:

```
POST /<existing_index>/_shrink/<new_index>
```

Чтобы инициировать операцию сжатия, необходимо составить запрос POST, указав в нем имя существующего индекса, который требуется сжать. Затем добавить ключевое слово `_shrink` в качестве первого сегмента URI и имя нового уменьшенного индекса.

Прежде чем проводить сжатие, необходимо убедиться, что выполняются следующие условия:

- **Индекс должен быть доступен только для чтения:** для сжатия индекса необходимо установить флаг `read-only` в `true`.
- **Все копии шарда должны находиться на одном узле:** копия каждого хранилища в индексе должна находиться на том же узле, на котором выполняется операция сжатия.
- **Состояние кластера должно быть зеленым:** в нем выделены все первичные шарды и реплики и кластер работает должным образом.

Когда эти условия выполнены, можно приступать к операции сжатия. API сжатия индекса также поддерживает несколько параметров запроса, включая следующие:

- `wait_for_active_shards` — задает количество активных шардов, необходимое для ответа API. Он позволяет контролировать время ожидания выделения шардов во время сжатия.
- `timeout` — задает максимальное время операции сжатия. Если операция превысит указанное значение тайм-аута, API вернет ответ.
- `master_timeout` — определяет максимальное время ожидания ответа от главного узла. Если главный узел не отвечает в течение указанного тайм-аута, API возвращает ошибку.

Кроме того, API сжатия индекса поддерживает тело запроса, в котором можно указать такие параметры, как псевдонимы и настройки, чтобы устанавливать псевдонимы индексов или изменять настройки индекса во время сжатия.

Важно отметить, что сжатие индекса — сложная операция и ее следует выполнять с осторожностью. Прежде чем начинать сжатие, необходимо его правильно спланировать и учесть его влияние на данные и ресурсы кластера.

Заморозка индекса

Чтобы заморозить индекс в Elasticsearch, можно воспользоваться API заморозки индекса. Заморозка снимает с кластера накладные расходы на обработку

индекса и делает его доступным только для чтения, блокируя любые операции записи. Замороженные индексы занимают минимум памяти, поскольку в ней хранятся только метаданные замороженного индекса.

Чтобы заморозить индекс, можно использовать следующую команду:

```
POST /<index_name>/_freeze
```

Чтобы заморозить определенный индекс, необходимо составить запрос POST, указав в нем имя индекса, а также URL-параметр `_freeze`.

Важно отметить, что заморозка индекса подразумевает следующие последствия:

- **Только для чтения:** после заморозки индекс становится доступным только для чтения, и никакие операции записи в него невозможны. Это обеспечивает неизменность замороженных данных.
- **Сокращение занимаемой памяти:** замороженные индексы занимают значительно меньше памяти, поскольку в ней сохраняются только необходимые метаданные, что минимизирует потребление ресурсов.

API заморозки индекса не требует дополнительных параметров. После успешного выполнения операции индекс будет заморожен и может использоваться только для чтения. При попытке записи в замороженный индекс будет выдано исключение.

Заморозка индексов может быть полезна в сценариях, когда данные статичны или являются историческими и не требуют дальнейшего изменения. Заморозив такие индексы, можно оптимизировать использование ресурсов и повысить производительность запросов для рабочих нагрузок с интенсивным чтением.

Обратите внимание, что API заморозки индексов доступен, начиная с версии Elasticsearch 7.12.

Разморозка индекса

Чтобы разморозить замороженный индекс в Elasticsearch, можно использовать API разморозки индекса. Разморозка делает индекс снова доступным для записи после выполнения стандартного восстановления. Чтобы разморозить индекс, используется следующая команда:

```
POST /<index_name>/_unfreeze
```

Чтобы разморозить указанный индекс, необходимо составить запрос POST, указав в нем имя замороженного индекса вместе с URL-параметром `_unfreeze`.

После успешного завершения разморозки индекс снова становится доступным для записи. Процесс восстановления гарантирует, что индекс полностью функционирует и готов принимать новые данные.

Важно отметить, что разморозка возвращает индекс в исходное состояние, позволяя выполнять операции чтения и записи. Однако она приводит к увеличению потребления памяти и ресурсов по сравнению с тем, когда индекс был заморожен. Поэтому рекомендуется размораживать индексы только при необходимости и, соответственно, контролировать ресурсы кластера.

API разморозки индекса не требует дополнительных параметров. После выполнения команды `_unfreeze` индекс перейдет из замороженного состояния в нормальное и станет доступным для записи новых данных.

ПРИМЕЧАНИЕ Обратите внимание, что API разморозки индекса доступен, начиная с версии Elasticsearch 7.12.

Разделение индекса

Чтобы разделить (split) индекс в Elasticsearch, можно использовать API разделения индекса, который увеличивает количество первичных шардов и распределяет существующие данные по новым шардам. Следующая команда служит для разделения индекса:

```
POST /<index_name>/_split/<new_index>
```

Мы составляем запрос POST и указываем имя существующего индекса, который хотим разделить. Далее указываем ключевое слово `_split` в URL, а затем — имя нового индекса, который будет создан в процессе разделения.

Вот пример разделения индекса `newindex`:

```
1. POST /newindex/_split/split-newindex
2. {
3.   "settings": {
4.     "index.number_of_shards": 2
5.   }
6. }
```

В этом примере `newindex` разбивается на новый индекс `split-newindex`, имеющий два первичных шарда. После выполнения команды получим следующий ответ:

```
1. {
2.   "acknowledged" : true,
3.   "shards_acknowledged" : true,
4.   "index" : "split-newindex"
5. }
```

Этот ответ указывает, что операция разделения подтверждена и новый индекс `split-newindex` успешно создан.

API разделения индекса поддерживает набор параметров запроса, таких как `wait_for_active_shards`, `timeout` и `master_timeout`. Эти параметры позволяют настраивать поведение операции разделения. Кроме того, API поддерживает тело запроса, которое может включать псевдонимы и параметры настройки, чтобы устанавливать псевдонимы индексов или изменять параметры индекса в процессе разделения.

Разделение индекса может быть ресурсоемкой операцией, особенно если речь идет о больших индексах. Поэтому важно учитывать доступные ресурсы и мощность кластера, прежде чем выполнять разделение.

Клонирование индекса

Чтобы клонировать индекс в Elasticsearch, можно использовать API клонирования индекса. Структура этого API выглядит так:

```
POST /<index_name>/_clone/<target_index_name>
```

Выражение выше представляет собой запрос POST с указанием имени индекса, который требуется клонировать. Далее в URL указывается ключевое слово `_clone`, за которым следует имя целевого индекса.

Прежде чем создавать клон индекса, необходимо убедиться, что выполняются следующие условия:

- Исходный индекс должен быть помечен как доступный только для чтения.
- Состояние кластера должно быть зеленым.

API клонирования работает следующим образом:

1. Создает целевой индекс, используя то же определение, что и исходный индекс.
2. Жестко связывает сегменты исходного индекса с целевым индексом. Если жесткая привязка не разрешена, все сегменты копируются из исходного индекса в целевой.
3. Восстанавливает целевой индекс путем его повторного открытия, аналогично процессу восстановления закрытого индекса.

Ниже представлен пример клонирования `newindex`:

1. Блокировка операций записи в исходный индекс:

```
1. PUT /testindex/_settings
2. {
```

```
3.   "settings": {  
4.     "index.blocks.write": true  
5.   }  
6. }
```

2. Создание индекса-клона с помощью следующей команды:

```
POST /testindex/_clone/newtestindex
```

3. После выполнения команды получим следующий ответ:

```
1. {  
2.   "acknowledged" : true,  
3.   "shards_acknowledged" : true,  
4.   "index" : "newtestindex"  
5. }
```

Ответ сообщает, что операция клонирования подтверждена и новый `newtestindex` успешно создан.

API клонирования индекса поддерживает различные параметры запроса, такие как `wait_for_active_shards`, `timeout` и `master_timeout`, которые позволяют настроить поведение операции клонирования. Кроме того, API поддерживает тело запроса, которое может включать псевдонимы и параметры настройки, чтобы устанавливать псевдонимы индексов или изменять параметры индекса в процессе клонирования.

Клонирование индекса может быть ресурсоемким, особенно для больших индексов, поэтому очень важно учитывать доступные ресурсы и мощность кластера, прежде чем выполнять эту операцию.

Переключение индекса

API переключения (`rollover`) индекса позволяет определить условия, на основании которых новому индексу может быть присвоен псевдоним текущего индекса. Эти условия задаются с помощью API переключения для управления размером, возрастом или количеством документов в индексе. Ниже приведен пример использования API переключения.

1. Создание начального индекса с псевдонимом:

```
1. PUT /new_roll_index-000001  
2. {  
3.   "aliases": {  
4.     "roll_alias": {  
5.       "is_write_index": true  
6.     }  
7.   }  
8. }
```

В этом примере мы создаем индекс `new_roll_index-000001` и присваиваем ему псевдоним `roll_alias`. Флаг `is_write_index` установлен в `true`, указывая, что этот индекс является текущим активным индексом записи.

2. Выполнение API переключения индекса с применением условий:

```

1. POST /roll_alias/_rollover
2. {
3.   "conditions": {
4.     "max_age": "7d",
5.     "max_docs": 1000,
6.     "max_size": "5gb"
7.   }
8. }
```

В этой команде мы задаем условия переключения. Условия включают максимальный возраст — 7 дней, максимальное количество документов — 1000 и максимальный размер индекса — 5 Гбайт. Если любое из этих условий выполняется, будет создан новый индекс и псевдоним будет переназначен новому индексу.

После выполнения API переключения выдается ответ, указывающий на успех или неудачу операции. Ответ содержит информацию о старом и новом индексе, о том, произошло ли переключение, и о состоянии заданных условий.

Важно отметить, что API переключения не является планировщиком. Он проверяет условия и осуществляет переключение, если условия выполняются во время вызова API. Если требуется автоматизировать процесс переключения на основе расписания или других критериев, используйте политики ILM в Elasticsearch.

API переключения индекса поддерживает различные параметры запроса, такие как `dry_run`, `include_type_name`, `wait_for_active_shards`, `timeout` и `master_timeout`, для дополнительной настройки переключения. Кроме того, API поддерживает тело запроса, которое может включать псевдонимы, условия, маппинг и параметры настройки, чтобы настраивать псевдонимы индексов и изменять параметры индекса в процессе переключения.

Обратите внимание, что если ответ содержит параметры `"acknowledged"` и `"shards_acknowledged"`, установленные в значение `false`, а также `false` для условий, это означает, что указанные условия не были выполнены и перенос не произошел.

Настройки индекса

В этом разделе мы рассмотрим API для изменения настроек индекса. Эти API включают обновление настроек индекса, получение настроек и их анализ.

Обновление настроек индекса

API обновления настроек индекса позволяет изменять настройки индекса в реальном времени. Этот API можно применять для изменения различных параметров, таких как количество реплик, интервал обновления и др. Вот пример использования API для изменения количества реплик:

```
1. PUT /newindex/_settings
2. {
3.   "index": {
4.     "number_of_replicas": 2
5.   }
6. }
```

В этом примере мы используем запрос PUT к конечной точке `_settings` индекса `newindex`. В теле запроса указываем желаемые настройки индекса, установив значение `"number_of_replicas"` в 2. Это обновит количество реплик для индекса.

Ответ будет содержать поле `"acknowledged"` со значением `true`; это означает, что запрос на обновление настроек индекса успешно подтвержден.

API обновления параметров индекса поддерживает несколько параметров запроса, которые можно использовать для настройки операции. Вот некоторые из часто используемых параметров:

- `allow_no_indices` — когда установлено значение `true`, запрос не будет выдавать ошибку, если указанный индекс не существует.
- `expand_wildcards` — указывает, необходимо ли расширить подстановочные выражения до открытых, закрытых или скрытых индексов.
- `flat_settings` — возвращает настройки в плоском файле, а не во вложенном.
- `ignore_unavailable` — когда установлено значение `true`, запрос не будет выдавать ошибку, если указанный индекс недоступен.
- `preserve_existing` — указывает, следует ли сохранять существующие настройки, которые не обновляются.
- `тайм-аут` — устанавливает тайм-аут для запроса.
- `master_timeout` — устанавливает тайм-аут ожидания ответа от главного узла.

Кроме того, тело запроса может содержать параметр `"settings"` для изменения определенных настроек индекса. Чтобы вернуть настройку к значению по умолчанию, можно использовать значение `null`. Например, чтобы вернуть параметр `"refresh_interval"` к значению по умолчанию, можно использовать следующую команду:

```
1. PUT /newindex/_settings
2. {
```

```

3.     "index": {
4.         "refresh_interval": null
5.     }
6. }
```

Она удалит пользовательскую настройку интервала обновления и вернет ее к значению по умолчанию.

Используя API обновления параметров индекса, можно динамически изменять параметры индекса в соответствии с требованиями без необходимости переиндексировать данные или создавать весь индекс заново.

Получение настроек индекса

API получения настроек позволяет получить настройки индекса. Для этого используется следующая команда:

```
GET /<index_name>/_settings
```

Заменив `<index_name>` на фактическое имя индекса, мы получим его настройки. Например, чтобы посмотреть настройки индекса "newindex", выполним следующую команду:

```
GET /newindex/_settings
```

Ответ будет содержать настройки указанного индекса. Пример ответа:

```

1. {
2.     "newindex": {
3.         "settings": {
4.             "index": {
5.                 "number_of_shards": "1",
6.                 "blocks": {
7.                     "read_only_allow_delete": "true"
8.                 },
9.                 "provided_name": "newindex",
10.                "creation_date": "1586021401751",
11.                "number_of_replicas": "2",
12.                "uuid": "68hXo41gR9Wzb5_U1AdXWQ",
13.                "version": {
14.                    "created": "7060199",
15.                    "upgraded": "7060299"
16.                }
17.            }
18.        }
19.    }
20. }
```

Ответ содержит сведения о настройках индекса, включая количество шардов, разрешение только на чтение, предоставленное имя, дату создания, количество реплик, UUID и номер версии.

API получения настроек поддерживает набор параметров запроса, которые можно использовать для настройки операции. Ниже перечислим некоторые из часто используемых параметров запроса:

- `allow_no_indices` — когда установлено значение `true`, запрос не будет выдавать ошибку, если указанный индекс не существует.
- `expand_wildcards` — указывает, необходимо ли расширить подстановочные выражения до открытых, закрытых или скрытых индексов.
- `flat_settings` — возвращает настройки в плоском файле, а не во вложенном.
- `include_defaults` — когда установлено значение `true`, в ответ включаются все настройки по умолчанию.
- `ignore_unavailable` — когда установлено значение `true`, запрос не будет выдавать ошибку, если указанный индекс недоступен.
- `local` — когда установлено значение `true`, настройки будут получены только с локального узла.
- `master_timeout` — устанавливает тайм-аут ожидания ответа от главного узла.

С помощью API получения настроек можно получить текущие настройки индекса, чтобы просмотреть и проверить конфигурацию имеющихся индексов Elasticsearch.

УПРАВЛЕНИЕ ШАБЛОНАМИ ИНДЕКСОВ ELASTICSEARCH

Среди шаблонов индексов можно определить тот, который будет автоматически применен при создании нового индекса. Шаблон содержит настройки и маппинг индекса, а также простой паттерн управления применением шаблона к новому индексу. Существуют различные API для управления шаблоном индекса.

Создание шаблона индекса

Чтобы создать или обновить шаблон индекса, можно использовать API шаблона индекса с запросом PUT. Вот пример выражения для создания шаблона индекса с именем "`test_template`":

```
1. PUT _template/test_template
2. {
3.   "index_patterns": ["an*", "test*"],
4.   "settings": {
5.     "number_of_shards": 1
```

```

6. },
7. "mappings": {
8.   "_source": {
9.     "enabled": false
10. },
11.   "properties": {
12.     "host_name": {
13.       "type": "keyword"
14.     },
15.     "created_at": {
16.       "type": "date",
17.       "format": "EEE MMM dd HH:mm:ss Z yyyy".
18.     }
19.   }
20. }
21. }

```

В приведенном выше примере шаблон индекса содержит следующие параметры:

- `index_patterns` — ссылается на массив шаблонов индексов для сопоставления с новыми именами индексов.
- `settings` — описывает настройки индекса, которые будут применяться к новым индексам, созданным с помощью этого шаблона.
- `mappings` — описывает маппинг, применяемый к новым индексам, созданным с помощью этого шаблона.

Используя это выражение, мы создали шаблон `test_template` с количеством шардов 1 и маппингом свойства `host_name` с типом ключевого слова и поля `created_at` с типом даты. Мы также указали `an*` и `test*` для поля `index_patterns`, чтобы шаблон можно было применить к любому индексу, начинающемуся с `an` или `test`. Предыдущее выражение выводит следующий ответ:

```

1. {
2.   "acknowledged" : true
3. }

```

Если значением поля `"acknowledged"` является `true`, это указывает, что шаблон индекса создан успешно.

Запрос PUT API шаблона индекса также поддерживает дополнительные параметры, такие как `create`, `order` и `master_timeout`. Эти параметры позволяют управлять поведением операции создания/обновления шаблона индекса.

Шаблон индекса работает только во время создания индекса, поэтому изменения в шаблоне не повлияют на индекс, созданный ранее. В шаблоне также можно использовать блочные комментарии `/* */` в стиле С.

Получение шаблона индекса

Чтобы получить информацию о шаблоне индекса, можно использовать API шаблона индекса с запросом `GET`. Вот пример команды для получения сведений о шаблоне индекса `test_template`:

```
GET /_template/test_template
```

Выполнив эту команду, получим следующий ответ:

```
1. {
2.   "test_template" : {
3.     "order" : 0,
4.     "index_patterns" : [
5.       "an*",
6.       "test*"
7.     ],
8.     "settings" : {
9.       "index" : {
10.         "number_of_shards" : "1"
11.       }
12.     },
13.     "mappings" : {
14.       "_source" : {
15.         "enabled" : false
16.       },
17.       "properties" : {
18.         "created_at" : {
19.           "format" : "EEE MMM dd HH:mm:ss Z yyyy",
20.           "type" : "data"
21.         },
22.         "host_name" : {
23.           "type" : "keyword"
24.         }
25.       }
26.     },
27.     "aliases" : { }
28.   }
29. }
```

Ответ будет содержать подробную информацию об указанном шаблоне индекса, включая паттерны индекса, настройки, маппинг и псевдонимы, связанные с этим шаблоном.

Удаление шаблона индекса

Чтобы удалить шаблон индекса, можно воспользоваться API шаблона индекса с запросом `DELETE`. Например, чтобы удалить шаблон индекса `test_template`, можно выполнить следующую команду:

```
DELETE /_template/test_template
```

Эта команда удаляет шаблон индекса `test_template`. Несколько шаблонов индексов можно удалить, указав список, разделенный запятыми, или подстановочное выражение.

УПРАВЛЕНИЕ ЖИЗНЕННЫМ ЦИКЛОМ ИНДЕКСА В ELASTICSEARCH

Index Lifecycle Management (ILM) в Elasticsearch позволяет определять правила и действия для управления жизненным циклом индексов. С помощью ILM можно автоматизировать такие действия, как перенос, принудительное слияние, заморозка, сжатие и удаление индексов на основе заранее определенных условий. Хотя политику жизненного цикла индексов можно создать с помощью API, это также легко сделать из пользовательского интерфейса Kibana. Для этого выполните следующие шаги.

1. Откройте интерфейс Kibana и нажмите ссылку **Management** (Управление) в левом меню.
2. На экране управления нажмите ссылку **Index Lifecycle Policies** (Политики жизненного цикла индекса). Откроется страница **Create an Index Lifecycle Policy** (Создание политики жизненного цикла индекса).
3. Укажите название политики.
4. В разделе **Hot phase** (Горячая фаза) включите опцию переключения, нажав на ползунок **Enable rollover** (Включить переключение). Установите нужные параметры, например максимальный размер индекса, максимальное количество документов и максимальный возраст для переноса.
5. Настройте теплую (**Warm**) и холодную (**Cold**) фазы в соответствии с требованиями. В теплой и холодной фазах можно указать такие действия, как заморозка или сжатие индекса.
6. Настройте фазу удаления (**Delete**) так, чтобы индекс удалялся через определенное количество дней после переноса.
7. После настройки нужных параметров нажмите кнопку **Save as New Policy** (Сохранить как новую политику), чтобы создать политику.

После создания политики ее можно просмотреть на странице списка политик жизненного цикла индексов. В правой части страницы указаны доступные действия с политикой, такие как добавление политики в шаблон индекса, а также там можно просмотреть индексы, связанные с политикой. Это позволяет связать политику с шаблоном индекса и автоматизировать управление индексами на основе заданной политики жизненного цикла.

ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели особенности управления индексами Elasticsearch, включая их создание, маппинг и основные операции на уровне индексов. Мы также рассмотрели использование REST API, специально разработанных для управления индексами Elasticsearch. Кроме того, мы обсудили важность шаблонов индексов и управление ими, а затем кратко рассмотрели управление жизненным циклом индекса.

В следующей главе мы сосредоточимся на применении функций поиска к данным, хранящимся в Elasticsearch. Начнем с поиска по URI и перейдем к поиску по телу, который позволит выполнять более сложные и специализированные запросы. Кроме того, мы изучим шаблоны поиска и мультипоиска, а также API мультипоиска, предоставляющие эффективные средства для одновременного выполнения нескольких поисковых запросов. Мы также изучим Explain API, который помогает понять, как оценивать запросы и их релевантность, и Profile API, позволяющий оценить характеристики производительности поисковых запросов.

ВОПРОСЫ

1. Опишите процесс создания индекса в Elasticsearch.
2. Как определить и создать маппинг индексов в Elasticsearch?
3. Как создать новый индекс в Elasticsearch и заполнить его документами?
4. Как выполнять такие операции, как закрытие, открытие и удаление индекса в Elasticsearch?
5. Как получить и изменить настройки индекса в Elasticsearch?
6. Перечислите шаги для получения существующего шаблона индекса или создания нового шаблона индекса в Elasticsearch.

ГЛАВА 7

Возможности поиска: освоение Query DSL и техник поиска

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели подробности управления индексами в Elasticsearch. Мы изучили создание индексов и процесс определения их маппинга. Кроме того, мы получили представление о том, как эффективно управлять индексами Elasticsearch с помощью различных операций. Мы подробно рассмотрели управление шаблонами индексов, а также ряд API на уровне индексов и разобрали, как управлять жизненным циклом индексов.

В этой главе мы перейдем к важной теме поиска данных в Elasticsearch. Начнем с изучения поиска по URI, а затем перейдем к поиску по телу запроса. Кроме того, разберем шаблоны поиска и мультипоиска, а также API мультипоиска. Также рассмотрим Explain API и Profile API. В условиях экспоненциального роста объема данных способность извлекать релевантную информацию из огромных источников данных приобретает первостепенное значение. Elasticsearch предлагает широкие возможности поиска для извлечения наиболее важной информации.

В Elasticsearch каждое поле документа индексируется в целях эффективного выполнения запросов по всем полям. Запросы могут обрабатывать как один индекс, так и несколько. Типы поиска также могут быть разными, например поиск по определенным полям, таким как возраст, идентификационный номер или контактный номер телефона; полнотекстовый поиск документов, соответствующих ключевому слову; либо поиск, сочетающий оба подхода в одном запросе. Изучение возможностей поиска в Elasticsearch мы начнем

с поиска по URI, который позволяет создавать запросы и напрямую получать результаты.

СТРУКТУРА

В этой главе:

- Поиск по URI
- Query DSL
- Фильтры и запросы
- Мульти поиск
- Шаблоны поиска и мультипоиска
- Explain API
- Profile API

ЦЕЛИ

Завершив чтение этой главы, вы получите полное представление о **предметноориентированном языке запросов (Query Domain Specific Language)** в Elasticsearch. Вы научитесь выполнять операции поиска по URI и по телу, эффективно используя возможности Elasticsearch по составлению запросов. Более того, вы станете разбираться в таких важнейших концепциях, как полнотекстовый поиск, соответствие фразе и нечеткий поиск, что позволит адаптировать запросы к нужным требованиям. Кроме того, вы освоите использование API для получения информации о выполнении запросов в целях точной настройки производительности и оптимизации поисковых запросов. Эта глава поможет вам уверенно и эффективно ориентироваться в функциональности Elasticsearch для работы с запросами.

ПОИСК ПО URI

Поиск по URI в Elasticsearch — это выполнение поисковых операций путем передачи параметров поиска непосредственно в **унифицированный идентификатор ресурса (URI)** HTTP-запроса. Это альтернатива выполнению поиска с использованием тела запроса DSL.

В ходе поиска по URI параметры поиска включаются в URL, что делает его удобным для выполнения быстрого и простого поиска без необходимости

построения сложных DSL-запросов. Параметры поиска добавляются к конечной точке как параметры строки запроса. Поиск по URI может быть двух типов: пустой поиск и поиск по полю.

Пустой поиск

Пустой поиск — это самый простой поиск, при котором не задается конкретный запрос и выводятся все документы по всем индексам.

Чтобы выполнить пустой поиск, используется следующий запрос:

```
GET /_search
```

Он вернет документы из всех доступных индексов Elasticsearch. Чтобы получить документы из определенного индекса, необходимо указать имя индекса перед конечной точкой `_search`. Например:

```
GET /userdetails/_search
```

Этот запрос вернет все документы из индекса `userdetails`. Чтобы получить документы из нескольких индексов, имена индексов необходимо передать через запятую:

```
GET userdetails,companies/_search
```

Указанная команда извлекает документы из индексов `userdetails` и `companies`. Кроме того, можно использовать подстановочные знаки для сопоставления имен индексов и получения документов из индексов, соответствующих критериям подстановочного знака. Например:

```
GET u*,m*/_search
```

По этому запросу будут получены документы из индексов, начинающихся с символов 'u' и 'm'. Поиск по URI обеспечивает быстрый и простой способ получения документов из одного или нескольких индексов или даже из всех индексов в кластере. Однако важно отметить, что он ограничен по сложности запросов и расширенным возможностям по сравнению с более мощным Query DSL. Для более сложных и гибких поисковых запросов рекомендуется использовать Query DSL и выполнять поиск по телу.

Поиск по полю

При поиске по URI в Elasticsearch можно передать поисковый запрос в URI в качестве параметра для выполнения поиска. Один из сценариев использования — быстрый поиск, когда нужно указать только поисковый термин

без других параметров. Рассмотрим, как работает поиск по URI, на примере индекса `userdetails`.

Предположим, у нас есть следующий документ в индексе `userdetails`:

```

1. {
2.   "_index": "userdetails",
3.   "_type": "_doc",
4.   "_id": "2",
5.   "_version": 1,
6.   "_seq_no": 3,
7.   "_primary_term": 1,
8.   "found": true,
9.   "_source": {
10.     "name": " Sophia",
11.     "gender": "female",
12.     "city": "Singapore",
13.     "age": 32
14.   }
15. }
```

Чтобы найти имя «**Sophia**» в индексе `userdetails`, нужен следующий запрос URI:

```
GET userdetails/_search?q=name:Sophia
```

В этом запросе мы указываем имя поля (`name`), за которым следует двоеточие (`:`) и поисковый термин (`Sophia`). Этот запрос вернет следующий результат:

```

1. {
2.   "took": 26,
3.   "timed_out": false,
4.   "_shards": {
5.     "total": 1,
6.     "successful": 1,
7.     "skipped": 0,
8.     "failed": 0
9.   },
10.  "hits": {
11.    "total": {
12.      "value": 1,
13.      "relation": "eq"
14.    },
15.    "max_score": 1.2039728,
16.    "hits": [
17.      {
18.        "_index": "userdetails",
19.        "_type": "_doc",
20.        "_id": "2",
21.        "_score": 1.2039728,
22.        "_source": {
```

```

23.           "name": "Sophia",
24.           "gender": "female",
25.           "city": "Singapore",
26.           "age": 32
27.       }
28.     ]
29.   ]
30. }
31. }
```

Полученный результат состоит из нескольких разделов. В поле `took` указано время в миллисекундах, которое потребовалось Elasticsearch для обработки запроса, а в поле `timed_out` — был ли тайм-аут. Раздел `_shards` содержит информацию о шардах, задействованных в поиске.

Раздел `hits` содержит фактические результаты поиска. Поле `total` показывает общее количество совпавших документов, а поле `max_score` — максимальное соответствие среди совпавших документов. Каждый документ в массиве `hits` содержит такие метаданные, как имя индекса (`_index`), тип документа (`_type`) и идентификатор документа (`_id`). Поле `_score` представляет собой оценку, рассчитанную Elasticsearch на основе релевантности документа поисковому запросу. Поле `_source` содержит фактические значения документов. Помимо базового поиска, поиск по URI позволяет передавать различные параметры для уточнения критерии, например, следующие:

- '`q`' — параметр `q` представляет собой строку запроса, в которой можно указать ключевое слово для поиска вместе с именем поля, например `filename:keyword`. Взгляните на следующее выражение:

```
GET.userdetails/_search?q=name:A*
```

Это выражение содержит параметр `q` с подстановочным знаком `*`, чтобы поиск в поле `name` выполнялся по всем именам, начинающимся с `A`.

- '`size`' — по умолчанию Elasticsearch выводит 10 записей, и это количество можно изменить, указав параметр `size`. См. выражение ниже:

```
GET.userdetails/_search?q=name:A*&size=3
```

В этом выражении параметр `size` ограничивает размер вывода до 3 вместо 10 по умолчанию.

- '`from`' — с помощью параметра `from` можно задать начальный индекс результата. По умолчанию Elasticsearch начинает с индекса 0:

```
GET.userdetails/_search?q=name:A*&from=2
```

В приведенном примере параметр `from` устанавливает начало поиска результатов с индекса 2, а не с индекса по умолчанию, который в Elasticsearch равен 0.

- 'sort' — к любому полю можно применить сортировку, указав имя поля вместе с типом сортировки; например `fieldname:asc` или `fieldname:desc` для сортировки имени поля `fieldname` по возрастанию или по убыванию. См. выражение ниже:

```
GET.userdetails/_search?q=name:A*&sort=age:asc
```

Оно сортирует результаты по возрасту пользователя в порядке возрастания. Вместо `asc` можно использовать `desc`, чтобы отсортировать результаты по убыванию.

- 'analyzer' — в поисковом запросе URI Elasticsearch также можно передать анализатор. Анализатор для запроса можно выбрать явно. См. следующее выражение:

```
GET.userdetails/_search?q=name:A*&analyzer=stop
```

Этот запрос содержит параметр `analyzer`, который установлен в значение `stop`. Таким образом, можно указать анализатор в поисковом запросе URI.

- 'explain' — с помощью этого параметра можно получить подробную информацию о том, как вычислялась оценка документа для каждого совпадения. См. следующее выражение:

```
GET.userdetails/_search?q=name:A*&explain=true
```

- Используя эту команду с параметром `explain`, получим следующий ответ:

```
1. {
2.     "_shard" : "[userdetails][0]",
3.     "_node" : "OsuawgbGQd2KXbWE3ENfEg",
4.     "_index" : "userdetails",
5.     "_type" : "_doc",
6.     "_id" : "4",
7.     "_score" : 1.0,
8.     "_source" : {
9.         "name" : "John",
10.        "gender" : "male",
11.        "city" : "London",
12.        "age" : 30
13.    },
14.    "_explanation" : {
15.        "value" : 1.0,
16.        "description" : "name:a*",
17.        "details" : [ ]
18.    }
19. }
```

Ответ содержит дополнительную информацию, такую как имена шардов и узлов, а также пояснение подсчета показателей.

QUERY DSL

В Elasticsearch тело запроса используется для построения запросов с помощью Query DSL. Query DSL обеспечивает удобство создания сложных поисковых запросов и запросов на агрегацию данных.

Чтобы продемонстрировать, как писать запрос в теле запроса, рассмотрим тот же поисковый запрос, который мы использовали в примере поиска по URI, но выраженный в формате тела запроса. Ниже приведен пример тела запроса для поиска документа с именем “*sophia*”:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "term": {
5.       "name": {
6.         "value": "sophia"
7.       }
8.     }
9.   }
10. }
```

В поле запроса *query* определяем тип запроса, в данном случае — *term*-запрос (точный запрос). В *term*-запросе указываем имя поля и соответствующее ему значение *sophia*, которое хотим найти. Ответ на данный запрос будет похож на тот, что был получен в примере поиска по URI:

```

1. {
2.   "took": 0,
3.   "timed_out": false,
4.   "_shards": {
5.     "total": 1,
6.     "successful": 1,
7.     "skipped": 0,
8.     "failed": 0
9.   },
10.  "hits": {
11.    "total": {
12.      "value": 1,
13.      "relation": "eq"
14.    },
15.    "max_score": 1.2039728,
16.    "hits": [
17.      {
18.        "_index": "userdetails",
19.        "_type": "_doc",
20.        "_id": "2",
21.        "_score": 1.2039728,
22.        "_source": {
```

```

23.         "name": "Sophia",
24.         "gender": "female",
25.         "city": "Singapore",
26.         "age": 32
27.     }
28.   }
29. ]
30. }
31. }
```

Структура ответа похожа на пример поиска по URI и содержит такую информацию, как время выполнения (`took`), информация о шардах (`_shards`) и результаты поиска (`hits`).

ПРИМЕЧАНИЕ В term-запросе термин приводится к нижнему регистру. Поэтому если в нашем примере искать имя “Sophia”, оно не будет соответствовать значению термина “sophia” из-за различия в регистре. Мы подробно рассмотрим типы запросов и их поведение в следующих разделах.

Использование тела запроса с Query DSL позволяет создавать более сложные запросы, комбинируя различные типы запросов, применяя фильтры и указывая параметры поиска. Это обеспечивает гибкость и выразительность поисковых запросов для конкретных потребностей.

ФИЛЬТРЫ И ЗАПРОСЫ

В Elasticsearch для поиска документов используются запросы и фильтры, но в разных целях. Для достижения желаемого поведения поиска важно понимать, когда следует использовать запрос, а когда фильтр. Давайте разберемся, в чем разница.

- Понять, что такое **запросы**, помогут следующие утверждения.
 - Запросы используются, когда нужно определить, насколько хорошо фраза запроса соответствует документу, и рассчитать показатель релевантности на основе этого соответствия.
 - Запросы подходят для полнотекстового поиска, когда требуется найти документы, соответствующие определенным критериям с разной степенью релевантности.
 - Результаты поиска сортируются по степени релевантности, причем наиболее релевантные документы отображаются первыми.
 - Оценка релевантности представлена метаполем `_score` в результатах поиска.

- Понять, что такое **фильтры**, помогут следующие утверждения.
 - Фильтры используются, когда требуется найти документы, которые точно соответствуют фразе запроса, без вычисления оценки релевантности.
 - Фильтры подходят для точных совпадений и в случае необходимости применения определенных условий, чтобы сузить результаты поиска.
 - Фильтры работают быстрее, чем запросы, поскольку не требуют вычисления оценки релевантности.
 - Результаты поиска не сортируются по релевантности, но могут быть отсортированы отдельно по другим параметрам.

Используя соответствующую комбинацию запросов и фильтров, можно точно настроить поведение поиска в соответствии с конкретными требованиями. Теперь рассмотрим опции поиска по телу запроса, такие как запросы, фильтры и сортировка.

Запрос

С помощью ключевого слова можно проводить поиск в теле запроса, указывая значение для поиска в имени поля, по которому необходимо провести поиск. Создадим запрос и посмотрим, как можно искать значения в полях.

Выше мы использовали term-запрос для поиска значения поля в нижнем регистре. Однако если нам требуется точное совпадение, мы можем использовать выражение `fieldname.keyword`, которое ищет неанализированное значение. См. следующий пример:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "term": {
5.       "name.keyword": {
6.         "value": "John"
7.       }
8.     }
9.   }
10. }
```

В этом запросе мы ищем точное значение поля `"name"`, используя поле `"name.keyword"`, а не просто `"name"`. Таким образом, мы ищем неанализированное значение вместо анализированного. Ответ на этот запрос будет таким:

```

1. {
2.   "took" : 0,
3.   "timed_out": false,
4.   "_shards" : {
```

```

5.     "total" : 1,
6.     "successful" : 1,
7.     "skipped" : 0,
8.     "failed" : 0
9.   },
10.  "hits" : [
11.    "total" : {
12.      "value" : 1,
13.      "relation" : "eq"
14.    },
15.    "max_score" : 1.2039728,
16.    "hits" : [
17.      {
18.        "_index" : "userdetails",
19.        "_type" : "_doc",
20.        "_id" : "4",
21.        "_score" : 1.2039728,
22.        "_source" : {
23.          "name" : "John",
24.          "gender" : "male",
25.          "city" : "London",
26.          "age" : 30
27.        }
28.      }
29.    ]
30.  }
31. }
```

В этом ответе мы получили результат, указав точное имя "John". Такой подход позволяет выполнять точный поиск данных, используя доступную функциональность запросов.

Типы запросов

В Elasticsearch есть три основных типа запросов, которые можно применять для поиска и получения данных: полнотекстовый поиск, запросы на уровне терминов и составные запросы. Каждый из них служит разным целям и может быть использован в зависимости от конкретного сценария.

- **Полнотекстовый поиск:** запросы служат для сопоставления искомого текста с текстовым полем. Перед выполнением поиска к полю применяется анализатор, который обрабатывает текст и преобразует его в более удобный для поиска формат. Этот процесс включает в себя такие операции, как токенизация, стемминг и удаление стоп-слов. Затем проанализированный текст сравнивается со значениями проанализированных полей; соответствующие результаты возвращаются в зависимости от степени сходства или релевантности. Полнотекстовый поиск особенно полезен, когда требуется найти

документы на основе их текстового содержания, например документы, содержащие определенные ключевые слова или фразы. Он позволяет осуществлять более гибкий поиск на естественном языке, с учетом языковых вариантов и форм слов.

- **Запросы на уровне терминов:** такие запросы, как следует из названия, работают на уровне точного термина. В отличие от полнотекстового поиска, предварительный анализ текста или значений полей не проводится. Эти запросы используются для сопоставления точного значения термина с полем. Значение термина никак не изменяется и не преобразуется, а сопоставление основывается на строгом равенстве. Запросы на уровне термина полезны, если требуется выполнить точный поиск и получить документы, в которых есть точное совпадение с определенным термином или значением. Такие запросы часто используются для фильтрации или для поиска неанализируемых полей, например идентификаторов, меток времени или перечисляемых значений.
- **Составные запросы:** они позволяют объединять несколько простых запросов для создания более сложных и эффективных. Для объединения простых запросов и определения желаемого поведения поиска в них используются такие логические операторы, как "must" (AND), "should" (OR) и "must_not" (NOT). Составные запросы полезны при построении сложных поисковых запросов, включающих несколько критериев или условий. Они обеспечивают гибкость выражения сложных поисковых требований и могут использоваться для создания расширенных функций поиска в приложениях.

Понимая и используя запросы этих типов, можно эффективно искать и извлекать данные из Elasticsearch в соответствии с потребностями и задачами.

Далее мы подробно рассмотрим каждый из них, чтобы понять, как их можно использовать в различных сценариях.

Полнотекстовый поиск

В полнотекстовых поисковых запросах анализируется текст поиска и значения полей и проводится их сопоставление на основе сходства. В Elasticsearch есть несколько вариантов полнотекстовых поисковых запросов:

- `match_all`
- `match`
- `match_phrase`
- `multi_match`
- `query_string`

match_all

Запрос `match_all` сопоставляет все документы в индексе и присваивает каждому документу оценку 1.0. Он полезен при поиске всех документов без указания поисковых терминов. Запрос можно выполнить следующим образом:

```
1. GET /_search
2. {
3.     "query": {
4.         "match_all": {}
5.     }
6. }
```

Кроме того, существует запрос `match_none`, который является противоположностью `match_all` и не включает ни одного документа. Его можно использовать для исключения всех документов из результатов поиска. Синтаксис запроса `match_none` выглядит так:

```
1. GET /_search
2. {
3.     "query": {
4.         "match_none": {}
5.     }
6. }
```

match

Запрос `match` ищет документы, которые соответствуют определенному значению в поле (запрос на соответствие). Имеющийся текст сначала анализируется, а затем сопоставляется со значениями полей. Запрос поддерживает различные типы данных, такие как текст, числа, булевые выражения и даты. Вот пример использования запроса `match` для поиска документов, содержащих имя "Sophia":

```
1. GET.userdetails/_search
2. {
3.     "query": {
4.         "match": {
5.             "name": {
6.                 "query": "Sophia"
7.             }
8.         }
9.     }
10. }
```

В этом выражении мы сопоставляем поле имени с запросом `Sophia`. В качестве ответа получим следующий документ:

```
1. {
2.     "_index" : "userdetails",
3.     "_type" : "_doc",
```

```

1.      "_id" : "3",
2.      "_score" : 1.2039728,
3.      "_source" : {
4.          "name" : "Sophia",
5.          "gender" : "female",
6.          "city" : "Singapore",
7.          "age" : 32
8.      }
9.  }
10. }
```

Запрос `match` возвращает документы, в которых поле `name` соответствует запросу `"Sophia"`. В ответе будет содержаться соответствующий документ с оценкой.

В запросе `match` можно указать оператор, например `"and"` или `"or"`. По умолчанию оператор установлен на `"or"`, это означает, что ответ будет содержать документы, соответствующие любому из терминов. Однако можно изменить оператор на `"and"`, чтобы получить документы, соответствующие всем терминам. Вот пример:

```

1. GET.userdetails/_search
2. {
3.     "query": {
4.         "match": {
5.             "name": {
6.                 "query": "Sophia Julian",
7.                 "operator": "and"
8.             }
9.         }
10.    }
11. }
```

В этом примере Elasticsearch будет искать документы, содержащие `"Sophia"` и `"Julian"` в поле `name`. Если оператор не указан, Elasticsearch по умолчанию использует оператор `"or"`.

match_phrase

Запрос `match_phrase` используется для получения документов, содержащих точное совпадение полного предложения или фразы, а не отдельных слов. При этом порядок слов в критериях поиска в найденных документах будет сохранен. Вот пример:

```

1. GET.userdetails/_search
2. {
3.     "query": {
4.         "match_phrase": {
5.             "name": {
6.                 "query": "Sophia Julian".
7.             }
8.         }
9.     }
10. }
```

```

8.      }
9.    }
10. }
```

В этом запросе мы ищем документы, в которых поле «имя» точно соответствует фразе "Sophia Julian". В результате будут возвращены только документы с именем "Sophia Julian". В ответе будет содержаться соответствующий документ с оценкой. Посмотрите на ответ для предыдущего запроса:

```

1. {
2.   "_index" : "userdetails",
3.   "_type" : "_doc",
4.   "_id" : "5",
5.   "_score" : 1.7770996,
6.   "_source" : {
7.     "name" : "Sophia Julian",
8.     "gender" : "female",
9.     "city" : "Singapore",
10.    "age" : 32
11.  }
12. }
```

Таким образом, запрос `match_phrase` позволяет искать точную фразу, а не отдельные слова и задавать нужный порядок слов в критериях поиска.

multi_match

Запрос `multi_match` обеспечивает поиск по нескольким полям документа. Это полезно, когда требуется сопоставить поисковый термин с несколькими полями. Вот пример использования запроса `multi_match`:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "multi_match": {
5.       "query": "Sophia Julian",
6.       "fields": ["name", "details"]
7.     }
8.   }
9. }
```

В этом запросе мы ищем документы, в которых поисковый термин "Sophia Julian" встречается в поле "name" или "details". Указав параметр поля в виде массива имен полей, можно проводить поиск по нескольким полям в одном запросе.

Запрос `multi_match` является гибким и позволяет искать по разным полям в зависимости от требований. Он упрощает процесс поиска термина по нескольким полям.

query_string

Запрос `query_string` предоставляет гибкий синтаксис, который разбирается парсером запросов. Он позволяет создавать сложные запросы с использованием таких операторов, как AND, OR и NOT. Текст в строке запроса анализируется перед сопоставлением с указанными полями. Запрос `query_string` поддерживает подстановочные символы и поиск по нескольким полям, что делает его мощным инструментом поиска в Elasticsearch.

Вот пример использования запроса `query_string`:

```
1. GET.userdetails/_search
2. {
3.   "query": {
4.     "query_string": {
5.       "query": "Sophia OR Julian",
6.       "default_field": "name"
7.     }
8.   }
9. }
```

Этот запрос ищет документы, в которых в поле `"name"` встречается либо `"Sophia"`, либо `"Julian"`. Оператор `OR` указывает, что может совпадать любой из терминов. Стока запроса анализируется и сопоставляется с указанным полем. Также присутствует параметр `"default_field"`, который указывает, что если в строке запроса не указано поле, то в качестве поля по умолчанию для сопоставления должно использоваться поле `"name"`.

Запросы `query_string` также могут быть более сложными и включать группы и несколько условий. Например:

```
1. GET.userdetails/_search
2. {
3.   "query": {
4.     "query_string": {
5.       "query": "( Sophia AND Julian) OR John",
6.       "default_field": "name"
7.     }
8.   }
9. }
```

В этом примере мы сгруппировали условия, в которых имя должно быть либо `"Sophia Julian"`, либо `"John"`. Круглые скобки указывают на группировку условий и определяют порядок их обработки.

Кроме того, запрос `query_string` позволяет проводить поиск по нескольким полям с помощью параметра `"fields"`. Можно указать массив имен полей для поиска по нескольким полям, например, так:

```
"fields": ["name", "details"]
```

Используя запрос `query_string`, можно выполнять полнотекстовые поисковые запросы с расширенными функциями, такими как операторы, группировка и поиск по нескольким полям. Теперь обсудим, как выполнять запросы на основе терминов.

Запросы на уровне терминов

В Elasticsearch запросы на уровне терминов используются для точного поиска по конкретным значениям терминов. Они обычно применяются в нетекстовых полях, таких как числовые значения, идентификаторы, логины или любые поля, где требуется точное соответствие. Рассмотрим различные варианты поиска по терминам:

- term-запрос;
- множественный term-запрос;
- запрос существования;
- запрос диапазона;
- нечеткий запрос;
- запрос с использованием подстановочного символа.

Term-запрос

Как мы уже говорили, term-запросы выполняют поиск по термину в заданном поле. Как правило, term-запросы используются для точных значений, таких как цена, возраст, идентификатор или логин. Таким образом, его можно использовать в основном для нетекстовых полей, а для текстовых полей он не рекомендуется. Для них можно использовать запросы на соответствие. Ниже показан пример term-запроса:

```
1. GET.userdetails/_search
2. {
3.   "query": {
4.     "term": {
5.       "age": {
6.         "value": 30
7.       }
8.     }
9.   }
10. }
```

Здесь мы ищем пользователей в возрасте 30 лет.

Множественный term-запрос

Множественный term-запрос позволяет искать документы с одним или несколькими точными терминами для заданного поля. Он похож на term-запрос, но поддерживает несколько значений. Вот пример:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "terms": {
5.       "age": [30, 39]
6.     }
7.   }
8. }
```

Этот запрос ищет пользователей в возрасте 30 или 39 лет.

Запрос существования

Используя запрос существования (`exists`), можно получить документы, содержащие искомое поле. Поскольку Elasticsearch не имеет схемы, может оказаться, что поле существует не во всех документах. Следующее выражение используется для поиска поля, если оно содержится в документе:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "exists": {
5.       "field": "gender"
6.     }
7.   }
8. }
```

Этот запрос получает документы, в которых существует поле `"gender"`. Запрос полезен для поиска документов, в которых имеется определенное поле.

Запросы на уровне терминов обеспечивают точное сопоставление значений в конкретных полях. Они особенно полезны для нетекстовых полей или сценариев, где требуется точное совпадение.

Запрос диапазона

Запрос диапазона (`range`) позволяет получить документы со значениями полей в указанном диапазоне. Он полезен для поиска документов в определенном числовом диапазоне, диапазоне дат или любом сортируемом диапазоне значений. Вот пример:

```

1. GET.userdetails/_search
2. {
```

```

3.   "query": {
4.     "range": {
5.       "age": {
6.         "gte": 30,
7.         "lte": 35
8.       }
9.     }
10.   }
11. }
```

Это выражение представляет собой запрос диапазона, получающий документы, в которых возраст в поле "age" находится в диапазоне от 30 до 35 лет (включительно). Запрос диапазона можно применить и к другим полям, таким как цена, дата или количество.

Нечеткий запрос

Нечеткий запрос (*fuzzy*) позволяет получить документы, содержащие термин, схожий с поисковым, на основе расстояния Левенштейна. Расстояние Левенштейна измеряет разницу между двумя последовательностями и определяется как количество изменений, необходимых для преобразования одного термина в другой. Вот пример нечеткого запроса:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "fuzzy": {
5.       "name": {
6.         "value": "joh"
7.       }
8.     }
9.   }
10. }
```

В этом запросе мы ищем документы, в которых поле "name" соответствует "joh" в пределах расстояния Левенштейна. Нечеткий запрос может найти совпадения даже при изменении одного символа, такого как замена, добавление, удаление или транспонирование. Ответ будет содержать документы, имя которых близко к поисковому термину, например «John». Выполнив команду выше, получим следующий ответ:

```

1. {
2.   "_index" : "userdetails",
3.   "_type" : "_doc",
4.   "_id" : "4",
5.   "_score" : 0.9918202,
6.   "_source" : {
7.     "name" : "John",
8.     "gender" : "male",
```

```

9.           "city" : "London",
10.          "age" : 30
11.        }
12.      }
```

Запрос возвращает ответ с именем `John`, поскольку оно соответствует поисковому слову `joh`. Так строятся нечеткие запросы, соответствующие заданным требованиям.

Запрос с использованием подстановочного символа

Запрос с подстановочным символом (`wildcard`) позволяет искать документы, соответствующие определенному шаблону. Операторы подстановочных символов можно использовать для поиска нуля или более символов. Вот пример такого запроса:

```

1. GET.userdetails/_search
2. {
3.   "query": {
4.     "wildcard": {
5.       "name": {
6.         "value": "s*a"
7.       }
8.     }
9.   }
10. }
```

В этом запросе мы ищем документы, в которых поле `"name"` начинается с `"s"` и заканчивается `"a"`. Подстановочный символ `"*"` используется для поиска любых символов между ними. Ответ будет содержать документы с именем `"Sophia"`. Выполнив запрос выше, получим следующий ответ:

```

1. {
2.   "_index": "userdetails",
3.   "_type": "_doc",
4.   "_id": "3",
5.   "_score": 1.0,
6.   "_source": {
7.     "name": " Sophia ",
8.     "gender": "female",
9.     "city": "Singapore",
10.    "age": 32
11.  }
12. }
```

Такой результат получился, поскольку в запросе был указан подстановочный символ `s*a`. Можно скорректировать запрос в соответствии с требованиями, например опустить последний символ, чтобы получить все имена, начинающиеся

с символа `s`. Таким образом, можно изменять запросы с подстановочными символами и получать документы по заданному шаблону.

Сложные запросы

В Elasticsearch можно выполнять сложные поисковые операции, комбинируя различные условия запросов. Эти условия можно разделить на два типа: листовые (leaf clauses) и составные (compound clauses). Листовые условия просты и предполагают поиск ключевого слова в индексе. Составные условия, в отличие от них, объединяют несколько условий для более сложного поиска.

Булевый запрос

Булевый запрос позволяет объединить несколько запросов с помощью булевых операторов, чтобы найти соответствующие документы. В булевых запросах есть несколько типов вхождений:

- `"must"`: соответствующие документы должны удовлетворять условию.
- `"should"`: соответствующие документы могут удовлетворять условию.
- `"must_not"`: соответствующие документы не должны удовлетворять условию.
- `"filter"`: условие должно присутствовать в соответствующих документах. Разница в том, что `"must"` влияет на оценку степени соответствия, а `"filter"` — нет.

Чтобы объединить условия в булевый запрос, используется ключевое слово `"bool"`. Вот пример:

```
1. POST _search
2. {
3.   "query": {
4.     "bool": {
5.       "must": {
6.         "term": { "name": "sophia" }
7.       },
8.       "must_not": {
9.         "range": { "age": { "gte": 10, "lte": 20 } }
10.      },
11.      "should": [
12.        { "term": { "gender": "female" } }
13.      ]
14.    }
15.  }
16. }
```

Сложный запрос в примере выше получает документы с именем "sophia", полом "female" и возрастом не в диапазоне от 10 до 20 лет. Комбинируя различные условия с помощью булевых операторов, можно создавать сложные запросы.

После выполнения приведенного выше запроса получим ответ, содержащий соответствующие документы. Вот пример ответа:

```

1. {
2.   "взял": 5,
3.   "timed_out": false,
4.   "_shards": {
5.     "total": 27,
6.     "successful": 27,
7.     "skipped": 0,
8.     "failed": 0
9.   },
10.  "hits": {
11.    "total": {
12.      "value": 1,
13.      "relation": "eq"
14.    },
15.    "max_score": 1.5606477,
16.    "hits": [
17.      {
18.        "_index": "userdetails",
19.        "_type": "_doc",
20.        "_id": "3",
21.        "_score": 1.5606477,
22.        "_source": {
23.          "name": "Sophia ",
24.          "gender": "female",
25.          "city": "Singapore",
26.          "age": 32
27.        }
28.      }
29.    ]
30.  }
31. }
```

Ответ содержит такую информацию, как время выполнения запроса (поле "took"), сведения о выполнении на уровне шардов (поле "_shards") и совпадшие документы (поле "hits"). Каждое совпадение содержит индекс, тип документа, идентификатор, оценку релевантности и фактические пары «поле — значение» документа.

Таким образом, используя составные запросы и булевые операторы, можно выполнять расширенный поиск в Elasticsearch и получать соответствующие запросу документы и сопутствующую информацию.

Запрос с усилением веса

Запрос с усилением веса (*boosting*) в Elasticsearch позволяет получить документы, соответствующие положительному условию, и снизить оценку релевантности, если они также соответствуют отрицательному условию. В запросе с усилением можно задать положительные и отрицательные критерии соответствия, чтобы повлиять на оценку релевантности. Вот пример:

```

1. GET userdetails/_search
2. {
3.     "query": {
4.         "boosting" : {
5.             "positive" : {
6.                 "term" : {
7.                     "name": "sophia"
8.                 }
9.             },
10.            "negative" : {
11.                "term" : {
12.                    "city" : "London"
13.                }
14.            },
15.            "negative_boost" : 0,5
16.        }
17.    }
18. }
```

Здесь определены положительное и отрицательное условия. Положительное условие ищет документы с именем "*sophia*", а отрицательное влияет на оценку документов, где город — "*London*". Параметр "*negative_boost*" задает вес или влияние отрицательного условия на оценку.

Ответ на такой запрос будет содержать соответствующие документы. Вот пример ответа:

```

1. "hits": [
2.   {
3.     "_index": "userdetails",
4.     "_type": "_doc",
5.     "_id": "3",
6.     "_score": 0.9395274,
7.     "_source": {
8.       "name": "Sophia",
9.       "gender": "female",
10.      "city": "Singapore",
11.      "age": 32
12.    }
13.  },
14.  {
15.    "_index": "userdetails",
16.    "_type": "_doc",
```

```

17.     "_id": "5",
18.     "_score": 0.68786836,
19.     "_source": {
20.       "name": "Sophia Julian",
21.       "gender": "female",
22.       "city": "London",
23.       "age": 32
24.     }
25.   }
26. ]

```

В ответе видим, что документ с городом "London" имеет оценку **0.687**, а документ с городом "Singapore" — более высокую оценку **0.939**. Отрицательное усиление влияет на оценку, снижая степень релевантности документа, соответствующего отрицательному условию.

Используя запрос **boosting**, можно точно настроить оценку релевантности документов на основе положительных и отрицательных критериев соответствия, чтобы получить желаемые результаты.

МУЛЬТИПОИСК

В Elasticsearch можно проводить множественный поиск с помощью API или шаблона мультипоиска. Мультипоиск позволяет отправить несколько поисковых запросов в одном запросе API, делая поиск по разным индексам эффективнее. Структура запроса мультипоиска включает заголовок и тело, повторяющиеся для каждого поискового запроса. Она выглядит так:

```

1. header\n
2. body\n
3. header\n
4. body\n

```

Используя такую структуру, можно объединять несколько запросов. Теперь посмотрим, как это сделать с помощью API и шаблона мультипоиска.

API мультипоиска

Используя API мультипоиска, можно выполнить несколько операций поиска сразу в одном запросе API. Рассмотрим пример:

```

1. GET userdetails/_msearch
2. {}
3. {"query": {"match": {"имя": "sophia"}}
4. {"index": "kibana_sample_data_ecommerce"}
5. {"query": {"match": {"customer_first_name": "George"}}}

```

В этом примере мы инициируем в Elasticsearch мультипоиск по индексу "userdetails" с помощью запроса 'GET' к конечной точке '_msearch'.

- Пустые скобки '{}' служат разделителем отдельных поисковых запросов в мультипоисковом запросе.
- Первый поисковый запрос задается в последующем теле: '{"query": {"match": {"имя": "sophia"}}}'. Он ищет документы в индексе "userdetails", в котором поле "name" совпадает с "sophia".
- Далее мы задаем новый поисковый запрос, указав индекс в заголовке: '{"index": "kibana_sample_data_ecommerce"}'. Это инструкция для Elasticsearch выполнить следующий поисковый запрос по индексу "kibana_sample_data_ecommerce".
- После заголовка мы определяем второй поисковый запрос в теле: '{"query": {"match": {"customer_first_name": "George"}}}'. Это целевой запрос для индекса "kibana_sample_data_ecommerce", и он ищет документы, в которых полю "customer_first_name" соответствует "George".

Elasticsearch обрабатывает каждый поисковый запрос в мультипоисковом запросе по отдельности. Ответ будет содержать результаты поиска по каждому запросу, обеспечивая полный набор соответствующих документов из разных индексов в рамках одного вызова API.

ШАБЛОНЫ ПОИСКА И МУЛЬТИПОИСКА

В Elasticsearch с помощью шаблонов можно определять повторно используемые поисковые запросы с динамическими параметрами. Эти шаблоны предварительно компилируются и хранятся на сервере. Во время выполнения шаблонам можно передавать определенные значения или параметры, что делает их адаптируемыми к различным сценариям поиска.

Шаблон поиска

Давайте разберем концепцию шаблонов поиска на примере.

- **Определение шаблона поиска.** Рассмотрим сценарий, в котором требуется найти документы в индексе "userdetails" на основе заданного возрастного диапазона. Для этого можно создать шаблон поиска.

```
1. POST _scripts/search_template
2. {
3.   "script": {
4.     "lang": "mustache",
```

```

5.     "source": {
6.       "query": {
7.         "range": {
8.           "age": {
9.             "gte": "{{min_age}}",
10.            "lte": "{{max_age}}"
11.          }
12.        }
13.      }
14.    }
15.  }
16. }
```

В этом примере мы определяем шаблон поиска с помощью POST-запроса к конечной точке '`_scripts/search_template`'. Шаблон написан на языке шаблонов Mustache.

Шаблон состоит из запроса диапазона для фильтрации документов по полю `"age"`. Заполнители '`{{min_age}}`' и '`{{max_age}}`' указывают динамические параметры, которые будут предоставляться во время выполнения запроса.

- **Выполнение шаблона поиска.** Чтобы выполнить шаблон поиска и передать динамические параметры, используем запрос GET к конечной точке '`_search/template`'. Вот пример:

```

1. GET userdetails/_search/template
2. {
3.   "source": {
4.     "id": "search_template",
5.     "params": {
6.       "min_age": 30,
7.       "max_age": 40
8.     }
9.   }
10. }
```

В этом примере мы выполняем шаблон поиска, указав индекс (`"userdetails"`) и конечную точку '`_search/template`'. Тело запроса содержит идентификатор источника шаблона (`"search_template"`) и значения параметров `min_age` и `max_age` (30 и 40 соответственно).

Elasticsearch обрабатывает шаблон поиска, заменяя заполнители на значения параметров. Затем он выполняет результирующий поисковый запрос по индексу `"userdetails"` и возвращает соответствующие документы в указанном возрастном диапазоне.

Шаблон мультипоиска

Elasticsearch также поддерживает шаблоны мультипоиска. Это позволяет выполнять эффективный поиск по разным шаблонам в одном запросе API. Структура запроса шаблона мультипоиска выглядит следующим образом:

1. {{header}}
2. {{body}}
3. {{header}}
4. {{body}}
5. ...

Рассмотрим пример использования шаблонов мультипоиска:

```
1. POST _msearch/template
2. { "index": "userdetails"}.
3. {"source": {"id": "search_template_1", "params": {"name": "John"}}}
4. { "index": "logs"}.
5. {"source": {"id": "search_template_2", "params": {"level": "error"}}}
```

В этом примере мы ведем поиск по шаблону мультипоиска, используя запрос 'POST' к конечной точке '_msearch/template'.

- Первый шаблон выполняется для индекса "userdetails". Он ссылается на идентификатор шаблона "search_template_1" и предоставляет значение параметра 'name' как "John".
- Далее мы указываем индекс "logs" для второго шаблона. Он использует идентификатор шаблона "search_template_2" и передает значение параметра 'level' как "error".

Elasticsearch обрабатывает каждый шаблон отдельно, заменяя заполнители на заданные значения параметров. Он выполняет результирующие поисковые запросы по соответствующим индексам и возвращает соответствующие результаты поиска для каждого шаблона.

Использование шаблонов мультипоиска позволяет объединить преимущество повторно используемых шаблонов поиска с эффективностью выполнения нескольких шаблонов в одном вызове API.

EXPLAIN API

API Explain в Elasticsearch позволяет получить подробную информацию о степени соответствия документов конкретному запросу. Указав в качестве

параметра идентификатор документа, можно проанализировать, почему тот или иной документ соответствует или не соответствует запросу. Исправим и улучшим вот такой пример:

```

1. GET userdetails/_explain/3
2. {
3.   "query": {
4.     "match": {
5.       "name": "sophia"
6.     }
7.   }
8. }
```

В приведенном выше запросе "3" представляет собой идентификатор документа, для которого необходимо понять процесс сопоставления. Выполнив этот запрос, мы получим детали сопоставления документа с заданным запросом. Ответ API будет содержать подробную информацию о факторах оценки и релевантности документа по отношению к запросу:

```

1. {
2.   "_index": "userdetails",
3.   "_type": "_doc",
4.   "_id": "3",
5.   "_score": 0.8943213,
6.   "matched": true,
7.   "explanation": {
8.     "value": 0.8943213,
9.     "description": "Weight(age:sophia in 1) [PerFieldSimilarity], result of:",
10.    "details": [
11.      {
12.        "value": 0.8943213,
13.        "description": "score(doc=1,freq=1.0 = termFreq=1.0\\n), computed as boost * idf * tf from:",
14.        "details": [
15.          {
16.            "value": 1.0,
17.            "description": "boost",
18.            "details": []
19.          },
20.          {
21.            "value": 1.0,
22.            "description": "idf(docFreq=1, maxDocs=1)".
23.          },
24.          {
25.            "value": 1.0,
26.            "description": "tf(termFreq=1.0)".
27.          }
28.        ]
29.      }
30.    ]
31.  }
32.}
```

```

29.      }
30.    ]
31.  }
32. }
```

В ответе содержится подробная информация о процессе оценки. Она включает индекс, тип, идентификатор документа и его оценку по отношению к запросу. Поле "matched" указывает, соответствует ли документ запросу. Поле "explanation" содержит подробное описание расчета оценки соответствия.

Поле "value" содержит общую оценку, а поле "description" — текстовое описание процесса ее расчета. В поле "details" подробно описываются параметры расчета, включая boost (усиление), **обратную частоту документов** (inverse document frequency, idf) и **частоту терминов** (term frequency, tf).

Рассмотрим основные элементы ответа.

- '_index', '_type', '_id' — эти поля содержат индекс, тип и идентификатор оцениваемого документа.
- '_score' — оценка соответствия документа запросу. Чем она выше, тем выше соответствие.
- 'matched' — значение 'true' или 'false' в поле 'matched' указывает, соответствует ли документ запросу.
- 'explanation' — в этом поле приводится подробное описание расчета оценки документа.
 - 'value' — поле value содержит общую оценку документа;
 - 'description' — содержит текстовое описание процесса расчета оценки. Оно содержит информацию о весе, поле и терминах запроса, участвующих в расчете;
 - 'details' — содержит подробную информацию о параметрах расчета оценки, в том числе boost, idf и tf.

Каждый элемент имеет собственные поля 'value' и 'description', объясняющие его вклад в общую оценку. Для большей детализации могут добавляться вложенные поля 'details'.

Анализируя объяснение, мы можем получить представление о том, как Elasticsearch рассчитывает оценку соответствия документа на основе таких факторов, как частота термина, обратная частота документа и значения boost. Эта информация важна для понимания того, почему конкретный документ занимает более или менее высокую или более низкую позицию в результатах поиска, и может помочь повысить релевантность и точность запросов.

Обратная частота документа и частота термина

Обратная частота документа (IDF) и частота термина (TF) — два важных понятия в области информационного поиска и интеллектуального анализа текстов. Они используются для измерения важности или релевантности терминов в коллекции документов.

Обратная частота документа

IDF — это метрика, которая количественно оценивает, насколько редко встречается термин в коллекции документов. Он рассчитывается как логарифм общего количества документов, деленного на количество документов, содержащих определенный термин. Формула для IDF выглядит следующим образом:

$$\text{IDF} = \log(N / DF)$$

где:

N = общее количество документов в коллекции;

DF = количество документов, содержащих термин.

IDF присваивает больший вес терминам, которые реже встречаются в коллекции документов, и помогает таким образом выявить слова, которые несут большую информационную ценность.

Частота термина

Частота термина (TF) измеряет, как часто встречается термин в конкретном документе. Она рассчитывается как количество вхождений термина, деленное на общее количество терминов в документе. Формула для TF выглядит следующим образом:

$$TF = (\text{Количество вхождений термина в документ}) / (\text{Общее количество терминов в документе})$$

Этот показатель помогает определить относительную важность термина в конкретном документе.

Комбинация TF и IDF часто используется в схеме взвешивания, называемой TF-IDF, которая призвана отразить важность термина в документе по отношению к тому, как часто он встречается во всей коллекции документов. TF-IDF широко используется в различных приложениях, таких как поиск документов, классификация текстов и информационно-поисковые системы для ранжирования и извлечения релевантных документов на основе их содержания.

PROFILE API

Profile API в Elasticsearch — это мощный инструмент для отладки и оптимизации поисковых запросов. Он предоставляет подробную информацию о времени выполнения отдельных компонентов операции поиска. Включив профилирование, вы сможете выявить узкие места и проблемы с производительностью, что позволит точно настроить запросы для повышения их эффективности.

Вот пример использования Profile API:

```

1. GET userdetails/_search
2. {
3.   "profile": true,
4.   "query": {
5.     "match": {
6.       "name": {
7.         "query": "sophia"
8.       }
9.     }
10.   }
11. }
```

В этом примере мы включаем профилирование, установив параметр `profile` в `true`. В демонстрационных целях мы задаем запрос на соответствие поля `name` со значением `"sophia"`. После выполнения команды выше получим подробности профилирования вместе с результатами запроса:

```

1.   "profile": {
2.     "shards": [
3.       {
4.         "id": "[OsuaWgbGQd2KXbWE3ENfEg][userdetails][0]",
5.         "searches": [
6.           {
7.             "query": [
8.               {
9.                 "type": "TermQuery",
10.                 "description": "name:sophia",
11.                 "time_in_nanos": 1354098,
12.                 "breakdown": {
13.                   "set_min_competitive_score_count": 0,
14.                   "match_count": 0,
15.                   "shallow_advance_count": 0,
16.                   "set_min_competitive_score": 0,
17.                   "next_doc": 7040,
18.                   "match": 0,
19.                   "next_doc_count": 2,
20.                   "score_count": 2,
```

```

21.          "compute_max_score_count": 0,
22.          "compute_max_score": 0,
23.          "advance": 9356,
24.          "advance_count": 2,
25.          "score": 17092,
26.          "build_scorer_count": 7,
27.          "create_weight": 100601,
28.          "shallow_advance": 0,
29.          "create_weight_count": 1,
30.          "build_scorer": 1219995
31.      }
32.    }
33.  ],
34.  "rewrite_time": 9183,
35.  "collector": [
36.    {
37.      "name": "SimpleTopScoreDocCollector",
38.      "reason": "search_top_hits",
39.      "time_in_nanos": 55470
40.    }
41.  ]
42.  },
43.  ],
44.  "aggregations": []
45. }
46. ]
47. }
48. }
```

После выполнения запроса с включенной функцией профилирования в ответе наряду с результатами запроса будут содержаться сведения о профилировании. Раздел профилирования содержит сведения о времени выполнения различных компонентов запроса, таких как разбивка запроса, время перезаписи, детали коллектора и агрегации.

Рассмотрим этот ответ и разберем его составляющие.

- **shards** — в этом разделе содержится информация о шардах, участвующих в операции поиска.
- **searches**: — этот раздел под каждым шардом содержит подробную информацию о выполнении запроса.
- **query** — этот раздел содержит разбивку на низкоуровневые запросы Lucene и статистику их выполнения.
- **rewrite_time** — данное поле содержит суммарное время переписывания запроса.

- **collector** — в этом разделе описываются коллекторы Lucene, используемые во время поиска, а также их имена, причины использования и время выполнения.
- **aggregations** — если в поиск были включены агрегации, в этом разделе будет содержаться информация об их выполнении.

Анализируя детали профилирования, можно получить представление о характеристиках производительности запроса, в том числе понять время выполнения различных компонентов запроса, выявить потенциальные узкие места и оптимизировать запросы для повышения производительности.

Profile API — это ценный инструмент оптимизации запросов и устранения неполадок, позволяющий точно настроить поисковые запросы на основе подробной информации о времени выполнения.

ЗАКЛЮЧЕНИЕ

В этой главе мы изучили различные параметры поиска данных в Elasticsearch. Мы начали с рассмотрения методов поиска по URI и по телу запроса, понимания их различий и сценариев использования. Мы разобрали концепции запросов и фильтров, изучили различные типы запросов, такие как полнотекстовый поиск, запросы на уровне терминов и составные запросы.

Далее мы перешли к мультипоиску и узнали об API и шаблоне мультипоиска. Эта функция позволяет объединять несколько поисковых запросов в одном и получать данные из нескольких индексов или документов за один раз.

Мы также узнали об Explain API, который предоставляет полезную информацию об оценке документов и объясняет, почему те или иные документы соответствуют или не соответствуют конкретным запросам. Это помогает лучше понять механизм расчета степени соответствия в Elasticsearch.

Кроме того, мы изучили Profile API, мощный инструмент отладки и оптимизации производительности. С помощью Profile API можно получить подробную информацию о времени выполнения отдельных компонентов поискового запроса, что поможет выявить и устраниить узкие места в производительности.

В следующей главе мы перейдем к геозапросам и сфере геопространственных данных. Мы узнаем о типах геоданных, о том, как создавать геопривязки, а также о том, как эффективно добавлять и искать геопространственные

данные в Elasticsearch. Поиск на основе геолокации открывает новые возможности для приложений, которые полагаются на функции, основанные на местоположении.

ВОПРОСЫ

1. Что такое пустой поиск?
2. Как выполнить поиск по полю URI в Elasticsearch?
3. Что отличает запрос от фильтра в Elasticsearch?
4. Чем запрос `match_phrase` отличается от запроса `match`?
5. Что такое нечеткий запрос? Объясните на примере.
6. Объясните концепцию запроса с использованием подстановочного символа и приведите пример.

ГЛАВА 8

Работа с геоданными в Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели различные методы поиска данных в Elasticsearch и изучили стратегии поиска по URI и телу запроса. Мы разобрали особенности запросов и фильтров, изучив такие аспекты, как полнотекстовый поиск, запросы на уровне терминов и составные запросы. Мы также изучили операции мультипоиска и ценные сведения, предлагаемые API Explain и Profile.

В этой главе мы переходим к управлению геопространственными данными. Эта функциональность Elasticsearch позволяет эффективно хранить и запрашивать геоданные и служит краеугольным камнем для поиска по местоположению, который является неотъемлемой частью современных приложений. Будь то поиск услуг поблизости или оптимизация логистики, Elasticsearch отлично работает с геоданными. Мы рассмотрим механизмы, лежащие в основе запросов в пределах заданного радиуса, сортировки по близости и агрегации данных по географическим атрибутам.

Эта глава служит руководством по использованию геопространственной функциональности Elasticsearch и дает полное представление о хранении и запросе геоданных. Поскольку сервисы, основанные на определении местоположения, развиваются в различных отраслях, освоение этой функциональности приобретает решающее значение.

СТРУКТУРА

В этой главе:

- Введение в геопространственный поиск
- Типы геоданных в Elasticsearch
- Данные типа геоточка
- Данные типа геоформа
- Геозапросы и фильтры DSL
- Сценарий использования
- Геоагрегация

ЦЕЛИ

Завершив чтение этой главы, вы научитесь разбираться в типах геоданных, создавать маппинг для геоданных, а также грамотно включать и искать геоданные в Elasticsearch. Эти знания позволят эффективно управлять геопространственной информацией, оптимизировав поиск по местоположению и таким образом обогатив ваши навыки использования различных возможностей Elasticsearch.

ВВЕДЕНИЕ В ГЕОПРОСТРАНСТВЕННЫЙ ПОИСК

Геопространственный поиск — это запрос и получение данных на основе их географического положения. Он включает в себя поиск и анализ данных, связанных с определенными географическими координатами или регионами. Геопространственный поиск необходим для широкого спектра приложений и отраслей, включая картографию, логистику, транспорт, недвижимость, социальные сети и многое другое.

С распространением устройств, имеющих функцию определения местоположения, и увеличением доступности данных о местоположении необходимость эффективного поиска и анализа геопространственной информации становится все более актуальной. Традиционные методы поиска, основанные только на текстовых данных, часто не могут удовлетворить запросы и требования, связанные с местоположением.

Elasticsearch — это мощный и масштабируемый поисково-аналитический движок, который обеспечивает надежную поддержку геопространственных данных. Он позволяет пользователям индексировать, хранить, искать и анализировать данные, привязанные к определенному местоположению на поверхности

земли. Объединив возможности полнотекстового поиска Elasticsearch с геопространственными функциями, пользователи могут выполнять сложный и эффективный геопространственный поиск.

К распространенным сценариям использования геопространственного поиска относятся:

- **Поиск по близости** — поиск точек интереса или объектов вблизи определенного места. Например, поиск близлежащих ресторанов, заправок или гостиниц.
- **Поиск по границам** — определение точек или областей в пределах заданной границы. Например, поиск недвижимости в пределах определенного города или региона.
- **Фильтрация на основе расстояния** — фильтрация данных на основе расстояния до контрольной точки. Например, поиск всех клиентов в определенном радиусе от магазина или фильтрация твитов в определенном географическом диапазоне.
- **Агрегация и кластеризация** — анализ и обобщение геопространственных данных на основе заданных критериев. Например, создание тепловых карт, определение хотспотов или кластеризация точек данных.

Elasticsearch предлагает геопространственные типы данных, такие как геоточки и геоформы, и поддерживает широкий спектр геопространственных запросов, включая запросы расстояния, граничные запросы, полигональные запросы и др. Эти запросы можно комбинировать с другими критериями поиска для создания мощных и гибких поисковых возможностей.

В следующих разделах мы подробно рассмотрим геопространственные возможности Elasticsearch, изучим, как индексировать геопространственные данные, выполнять различные геопространственные запросы и использовать расширенные функции для геопространственного анализа и визуализации.

ТИПЫ ГЕОДАННЫХ В ELASTICSEARCH

При работе с геопространственными данными в Elasticsearch необходимо явно определить маппинг для типов геоданных. Elasticsearch предоставляет два основных типа геоданных:

- **Геоточка** (geo-point) — этот тип данных используется для хранения данных о местоположении в виде пар широта — долгота. Он идеально подходит для представления конкретной точки на поверхности земли. Например, тип данных геоточка можно использовать для хранения координат ресторана или местоположения клиента.

- **Геоформа** (`geo-shape`) — этот тип данных предназначен для работы со сложными фигурами, такими как круги, многоугольники или другие пользовательские фигуры. Эти фигуры задаются в формате GeoJSON, который позволяет точно представлять границы и области. Геоформы позволяют хранить и запрашивать данные, связанные с регионами, границами или любыми другими фигурами в пространстве.

Чтобы хранить данные этих типов в Elasticsearch, необходимо задать соответствующий маппинг, который определяет поле как геоточку или геоформу. Такой маппинг предоставит Elasticsearch необходимую информацию для правильного индексирования и обработки геопространственных данных.

После сохранения геопространственных данных с ними можно выполнять различные поисковые и аналитические операции, используя мощные поисковые возможности Elasticsearch. Можно искать документы на определенном расстоянии от заданного места, находить точки данных в определенных границах, а также объединять и анализировать геопространственные данные по определенным критериям.

ДАННЫЕ ТИПА ГЕОТОЧКА

Геоточки в Elasticsearch предлагают несколько полезных функций для работы с информацией о местоположении. Вот некоторые основные возможности геоточек.

- **Поиск по диапазону** — позволяет легко находить точки на заданном расстоянии от центральной точки.
- **Пространственная фильтрация** — позволяет находить геоточки, которые попадают в заданную границу или полигон.
- **Агрегация на основе расстояния** — возможность объединять документы на основе их удаленности от центральной точки, что позволяет анализировать данные в пределах определенного расстояния.
- **Оценка релевантности** — расстояние между местом запроса и геоточкой может быть использовано для определения оценки релевантности документов.
- **Сортировка по расстоянию** — возможность сортировать результаты поиска на основе их близости к заданному месту.

Эти функции обеспечивают эффективный геопространственный поиск и анализ в Elasticsearch.

Создание маппинга

Чтобы явно определить маппинг для геопространственных данных в Elasticsearch, используем следующее выражение для типа данных `geo_point`:

```
1. PUT geo_index
2. {
3.   "mappings": {
4.     "properties": {
5.       "location": {
6.         "type": "geo_point"
7.       }
8.     }
9.   }
10. }
```

Выполнив команду выше, мы создадим маппинг для поля `location` как типа данных `geo_point` в индексе `geo_index`. В случае успешного выполнения Elasticsearch выдаст ответ:

```
1. {
2.   "acknowledged" : true,
3.   "shards_acknowledged" : true,
4.   "index" : "geo_index"
5. }
```

После создания маппинга можно приступать к сохранению документов, содержащих геопространственные данные, в индекс `geo_index`.

Сохранение геоточек

В Elasticsearch существует несколько способов сохранения данных геоточек, и можно выбрать любой из них в зависимости от предпочтений. Вот пять таких способов:

- **Object** (Объект) — сохранить данные геоточки как объект, например:

```
1. POST geo_index/_doc
2. {
3.   "text": "Данные геоточки с типом объект",
4.   "location": {
5.     "lat": 42.14,
6.     "lon": -73.46
7.   }
8. }
```

- **String** (Строка) — сохранить данные геоточки в виде строки, где широта и долгота разделены запятой, например:

```

1. POST geo_index/_doc
2. {
3.   "text": "Данные геоточки с типом строки",
4.   "location": "42.14,-73.46"
5. }
```

- **Geohash** (Геохеш) — сохранить данные геоточки в виде геохеша, который представляет собой короткую буквенно-цифровую строку, обозначающую комбинацию широты и долготы, например:

```

1. POST geo_index/_doc
2. {
3.   "text": "Данные геоточки с типом геохеш",
4.   "location": "gbsuv7ztq"
5. }
```

- **Array** (Массив) — сохранить данные геоточки в виде массива, например, с указанием долготы перед широтой:

```

1. POST geo_index/_doc
2. {
3.   "text": "Данные геоточки с типом массив",
4.   "location": [ -73.46, 42.14 ]
5. }
```

- **Point** (Точка) — сохранить данные геоточки, используя формат точек **Well-Known Text (WKT)**, в виде "POINT (longitude latitude)"¹, например:

```

1. POST geo_index/_doc
2. {
3.   "text": "Данные геоточки с типом точка",
4.   "location": "POINT (-73.46 42.14)"
5. }
```

Используя любой из этих способов, можно сохранить данные геоточек в Elasticsearch, чтобы выполнять различные запросы и операции, основанные на местоположении.

ДАННЫЕ ТИПА ГЕОФОРМА

Тип данных геоформа в Elasticsearch позволяет индексировать и искать произвольные геометрические фигуры, такие как прямоугольники, многоугольники и круги. Этот тип данных используется, когда требуется индексировать или искать фигуры, а не местоположение точек. Elasticsearch поддерживает различные типы геофигур:

¹ «POINT (longitude latitude)» — «ТОЧКА (долгота — широта)». — Примеч. ред.

- **Point (Точка)** — представляет собой одну географическую координату с указанием широты и долготы.
- **LineString (Линия)** — определяет произвольную линию, которая соединяет две или более точки, используя массив позиций.
- **Polygon (Многоугольник)** — представляет собой замкнутую фигуру, первая и последняя точки которой должны совпадать. Она определяется списком точек.
- **MultiPoint (Набор точек)** — содержит несколько связанных точек, которые не соединены между собой. Может быть представлена в виде массива.
- **MultiLineString (Набор линий)** — объединяет несколько объектов LineString и может быть представлена в виде массива различных LineString.
- **MultiPolygon (Набор многоугольников)** — состоит из нескольких многоугольников и представляется в виде массива.
- **GeometryCollection (Коллекция геообъектов)** — представляет собой коллекцию геометрических объектов в формате GeoJSON.
- **Envelope (Конверт)** — представляет собой фигуру с двумя точками — левой верхней и правой нижней точками ограничивающего прямоугольника.
- **Circle (Окружность)** — определяется центральной точкой и радиусом, где единицей измерения по умолчанию являются метры.

Эти типы геофигур позволяют хранить и обрабатывать сложные геометрические данные в Elasticsearch. В следующем разделе мы рассмотрим, как создавать маппинг для этих типов геофигур.

Создание маппинга

Чтобы работать с данными типа geo_shape в Elasticsearch, необходимо создать явный маппинг. Посмотрим, как создать маппинг для геоформы. Для этого используем следующее выражение:

```

1. PUT geoshape_index
2. {
3.   "mappings": {
4.     "properties": {
5.       "location": {
6.         "type": "geo_shape"
7.       }
8.     }
9.   }
10 }
```

Эта команда создает маппинг для типа данных, в частности, для поля "location" в индексе "geoshape_index". После выполнения команды вы получите следующий ответ:

```
1. {
2.   "acknowledged": true,
3.   "shards_acknowledged": true,
4.   "index": "geoshape_index"
5. }
```

Этот ответ подтверждает, что команда выполнена успешно и для поля "location" в индексе "geoshape_index" создан маппинг типа данных geo_shape. После создания маппинга можно приступать к сохранению документов в индексе и выполнению запросов geo_shape.

Сохранение данных геоточки

Геофигуры могут быть представлены в двух форматах: GeoJSON и WKT. Рассмотрим типы геофигур и способы сохранения документов в каждом из них.

Точка

Для типа Point (Точка) мы можем использовать формат GeoJSON или WKT. Чтобы сохранить документ в формате GeoJSON, выполните следующую команду:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "данные геоформы с типом точка",
4.   "location": {
5.     "type" : "point",
6.     "coordinates" : [-73.03, 42.89]
7.   }
8. }
```

Используя это выражение, мы можем создать документ с типом геоточки с помощью GeoJSON. Чтобы создать документ типа геоточки в формате WKT, нужно написать следующий код:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "данные геоформы с типом точка",
4.   "location": "POINT (-73.46 42.14)".
5. }
```

Эта команда сохраняет геоданные типа точка в формате WKT.

Линия

Линия, тип геофиры LineString, определяется двумя или более точками, представленными в виде массива. Чтобы сохранить документ LineString в формате GeoJSON, выполните следующую команду:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "Данные геоформы с типом LineString",
4.   "location": {
5.     "type": "linestring",
6.     "coordinates": [[-78.03653, 36.897676], [-78.009051, 36.889939]]
7.   }
8. }
```

Эта команда сохраняет объект LineString в формате GeoJSON. Сохранить строку LineString в формате WKT можно с помощью следующего выражения:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "Данные геоформы с типом LineString",
4.   "location": "LINESTRING (-77.03653 38.897676, -77.009051
38.889939)"
5. }
```

Многоугольник

Многоугольник (Polygon) — это замкнутая фигура, у которой первая и последняя точки совпадают. У многоугольника с n сторонами должно быть $n+1$ вершин. Форма многоугольника задается списком точек. Сохранить форму многоугольника с помощью GeoJSON можно, используя следующее выражение:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "Данные геоформы с типом многоугольник",
4.   "location": {
5.     "type": "polygon",
6.     "coordinates": [
7.       [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0],
8.        [100.0, 0.0]]
9.     ]
10.   }
11. }
```

Это выражение сохраняет многоугольник в формате GeoJSON. Чтобы сохранить его в формате WKT, используйте такой код:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "Данные геоформы с типом многоугольник",
```

```

4.     "location": "POLYGON ((100.0 0.0, 101.0 0.0, 101.0 1.0, 100.0 1.0,
      100.0 0.0))"
5. }
```

Чтобы создать многоугольник с отверстием, необходимо определить внешнюю и внутреннюю границы. Внешняя граница образует внешний слой, а внутренняя — отверстие. Чтобы создать многоугольник с отверстием в формате GeoJSON, выполните следующее:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "Многоугольник с отверстием в формате GeoJSON",
4.   "location": {
5.     "type": "polygon",
6.     "coordinates": [
7.       [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0],
     [100.0, 0.0]],
8.       [[100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8],
     [100.2, 0.2]]
9.     ]
10.   }
11. }
```

При этом будет создан многоугольник с отверстием в формате GeoJSON. Чтобы создать такой же многоугольник с отверстием в формате WKT, используйте выражение:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "Многоугольник с отверстием в формате WKT",
4.   "location": "POLYGON ((100.0 0.0, 101.0 0.0, 101.0 1.0, 100.0 1.0,
      100.0 0.0), (100.2 0.2, 100.8 0.2, 100.8 0.8, 100.2 0.8, 100.2 0.2))"
5. }
```

Эти фрагменты кода демонстрируют, как сохранить многоугольные фигуры с отверстиями и без них в форматах GeoJSON и WKT. Аналогичные подходы можно использовать и для других типов фигур.

Набор точек

Набор точек, MultiPoint, — это тип фигуры, представляющий собой набор связанных точек, которые не соединены между собой. Чтобы создать фигуру MultiPoint с помощью GeoJSON, выполните следующее:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "MultiPoint в GeoJSON",
4.   "location": {
```

```

5.      "type": "multipoint",
6.      "coordinates": [
7.          [102.0, 2.0], [103.0, 2.0]
8.      ]
9.  }
10. }
```

Код выше сохраняет форму MultiPoint в формате GeoJSON.

Чтобы сохранить форму MultiPoint в формате WKT, выполните:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "MultiPoint в формате WKT",
4.   "location": "MULTIPOINT (102.0 2.0, 103.0 2.0)".
5. }
```

Этот код сохраняет данные геоформы типа MultiPoint в формате WKT.

MultiLineString

MultiLineString – тип фигуры, состоящий из нескольких фигур LineString. Для представления MultiLineString используется массив LineString. Вот как сохранить MultiLineString в GeoJSON:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "MultiLineString в GeoJSON",
4.   "location": {
5.     "type": "multilinestring",
6.     "coordinates": [
7.       [ [102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0] ],
8.       [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0] ],
9.       [ [100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8] ]
10.     ]
11.   }
12. }
```

Если вы предпочитаете сохранить MultiLineString в формате WKT, выполните следующий код:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "MultiLineString в формате WKT",
4.   "location": "MULTILINESTRING ((102.0 2.0, 103.0 2.0, 103.0 3.0, 102.0
3.0), (100.0 0.0, 101.0 0.0, 101.0 1.0, 100.0 1.0), (100.2 0.2, 100.8 0.2,
100.8 0.8, 100.2 0.8))"
5. }
```

Набор многоугольников

MultiPolygon — это коллекция многоугольников. Чтобы создать фигуру MultiPolygon, можно использовать массив многоугольников в формате GeoJSON. Ниже приведен пример сохранения MultiPolygon в GeoJSON:

```

1. POST geoshape_index/_doc/12
2. {
3.   "text": "MultiPolygon using GeoJSON",
4.   "location": {
5.     "type": "multipolygon",
6.     "coordinates": [
7.       [ [[102.0, 2.0], [103.0, 2.0], [103.0, 3.0], [102.0, 3.0],
[102.0, 2.0]] ],
8.       [ [[100.0, 0.0], [101.0, 0.0], [101.0, 1.0], [100.0, 1.0],
[100.0, 0.0]],
9.         [ [100.2, 0.2], [100.8, 0.2], [100.8, 0.8], [100.2, 0.8],
[100.2, 0.2]] ]
10.      ]
11.    }
12.  }
```

Если вы предпочтаете сохранять MultiPolygon в формате WKT, выполните следующий код:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "MultiPolygon using WKT",
4.   "location": "MULTIPOLYGON (((102.0 2.0, 103.0 2.0, 103.0 3.0, 102.0 3.0,
102.0 2.0)), ((100.0 0.0, 101.0 0.0, 101.0 1.0, 100.0 1.0, 100.0 0.0),
(100.2 0.2, 100.8 0.2, 100.8 0.8, 100.2 0.8, 100.2 0.2)))"
5. }
```

Коллекция геообъектов

Тип GeometryCollection представляет собой набор геометрических объектов. Его можно сохранить как коллекцию геометрических объектов в формате GeoJSON. Ниже приведен пример сохранения коллекции GeometryCollection в GeoJSON:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "GeometryCollection using GeoJSON",
4.   "location": {
5.     "type": "geometrycollection",
6.     "geometries": [
7.       {
8.         "type": "point",
```

```

9.      "coordinates": [100.0, 0.0]
10.     },
11.     {
12.       "type": "linestring",
13.       "coordinates": [ [101.0, 0.0], [102.0, 1.0] ]
14.     }
15.   ]
16. }
17. }
```

Если вы предпочтаете сохранять коллекцию GeometryCollection в формате WKT, выполните следующий код:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "GeometryCollection using WKT",
4.   "location": "GEOMETRYCOLLECTION (POINT (100.0 0.0), LINESTRING
    (101.0 0.0, 102.0 1.0))"
5. }
```

Конверт

Форму Конверт (Envelope) можно представить, задав левую верхнюю и правую нижнюю точки. Граница Конверта определяется как [[minLon, maxLat], [maxLon, minLat]]. Чтобы представить форму Envelope в GeoJSON, используйте такой код:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "Конверт в формате GeoJSON",
4.   "location": {
5.     "type": "envelope",
6.     "coordinates": [ [100.0, 1.0], [101.0, 0.0] ]
7.   }
8. }
```

Если вы предпочтаете сохранять форму Envelope в формате WKT, выполните следующий код:

```

1. POST geoshape_index/_doc
2. {
3.   "text": "Конверт в формате WKT",
4.   "location": "BBOX (100.0, 102.0, 2.0, 0.0)".
5. }
```

В спецификации WKT форма Envelope сохраняется в следующем порядке: minLon, maxLon, maxLat, minLat.

Окружность

Форму Окружность (Circle) можно задать, указав центральную точку и радиус с единицами измерения. По умолчанию единицей измерения являются метры. Форма окружности не представлена в формате GeoJSON или WKT. Чтобы сохранить данные окружности, можно использовать следующий код:

```
1. POST geoshape_index/_doc
2. {
3.   "text": "Circle data",
4.   "location": {
5.     "type": "circle",
6.     "coordinates": [101.0, 1.0],
7.     "radius": "100m"
8.   }
9. }
```

ГЕОЗАПРОСЫ И ФИЛЬТРЫ DSL

Как было сказано выше, в Elasticsearch поддерживаются два типа геоданных: `geo_point` и `geo_shape`. Тип `geo_point` используется для хранения координат широты и долготы в виде одной точки, а тип `geo_shape` поддерживает различные формы, такие как линии, круги и многоугольники. Elasticsearch предоставляет несколько типов запросов для поиска геоданных:

- Запрос `geo_distance` — позволяет найти документы с геоточками, которые находятся на заданном расстоянии от центральной точки. Этот тип запроса полезен для поиска близлежащих объектов. Запрос включает такие параметры, как центральная точка, расстояние и единица измерения расстояния.
- Запрос `geo_polygon` — позволяет найти документы с геоточками в пределах заданного многоугольника. Многоугольник определяется набором координат, представляющих его вершины. Этот запрос полезен для поиска местоположений в пределах определенной области или границы.
- Запрос `geo_bounding_box` — помогает найти документы с геоточками, которые попадают в заданную прямоугольную область. Она определяется двумя точками, расположенными по диагонали — левым верхним и правым нижним углами прямоугольника. Это быстрый и эффективный способ фильтрации документов на основе их расположения в пределах ограничительной области.
- Запрос `geo_shape` — извлекает документы, формы которых пересекаются с указанной формой запроса. Для определения пересечения используется квадратное представление сетки маппинга `geo_shape`. Этот запрос универсален и может использоваться для поиска документов с различными формами, включая многоугольники, круги и линии.

Это основные типы геозапросов, доступные в Elasticsearch. Каждый тип запроса предлагает собственную функциональность для поиска геоданных на основе близости, содержания или пересечения. Вы можете выбрать подходящий тип запроса в зависимости от конкретного сценария использования и характера геоданных.

Запросы георасстояния

Запросы георасстояния используются, чтобы найти документы в определенном диапазоне от заданной геоточки. Например, если вам нужно найти банкоматы в радиусе 5 км от вашего местоположения, вы можете использовать запросы георасстояния. Код ниже показывает, как выполнить эти запросы:

```
1. GET /geo_index/_search
2. {
3.   "query": {
4.     "bool": {
5.       "must": {
6.         "match_all": {}
7.       },
8.       "filter": {
9.         "geo_distance": {
10.           "distance": "100km",
11.           "location": {
12.             "lat": 42,
13.             "lon": -73
14.           }
15.         }
16.       }
17.     }
18.   }
19. }
```

В этом примере мы задаем условие фильтрации, чтобы получить все документы в радиусе 100 км от заданных координат широты и долготы. Значение расстояния и координаты местоположения можно изменить в соответствии с конкретным сценарием использования.

Расстояние в запросах георасстояния поддерживает различные единицы измерения, например:

- **Миля:** mi или miles
- **Ярд:** yd или yards
- **Километр:** km или kilometers
- **Метр:** m или meters
- **Сантиметр:** cm или centimeters

- **Миллиметр:** mm или millimeters
- **Морская миля:** NM, nmi или nautical miles
- **Футы:** ft или feet
- **Дюймы:** in или inch

Для поиска георасстояний можно использовать любые единицы измерения, например, для поиска в диапазоне 100 км мы использовали km.

Запросы георасстояния основаны на типе поля `geo_point`, поэтому они поддерживают различные представления местоположения. Местоположение можно представлять в следующих форматах:

- **Lat и Lon как отдельные свойства** — в данном случае местоположение представляется с отдельными значениями свойств `lat` и `lon`. Пример:

```
1. "location": {  
2.   "lat": 42,  
3.   "lon": -73  
4. }
```

- **Lat Long в виде массива** — в данном случае местоположение представляется в виде массива. Как уже говорилось, при представлении массива `lon` записывается перед значением `lat`. Пример:

```
"location": [-73, 42]
```

- **Lat Long в виде строки** — местоположение представляется в виде строки. Пример:

```
"location": "42,-73"
```

- **Lat Long в формате Geohash** — Geohash позволяет удобно представлять местоположение в виде короткой буквенно-цифровой строки. Пример:

```
"location": "gbsuv7ztq"
```

Геополигональные запросы

С помощью геополигональных запросов можно получить документы в пределах многоугольника, заданного набором точек. Вот пример геополигонального запроса:

```
1. GET /_search  
2. {  
3.   "query": {  
4.     "bool": {  
5.       "must": {
```

```

6.      "match_all": {}
7.    },
8.    "filter": {
9.      "geo_polygon": {
10.        "location": {
11.          "points": [
12.            {"lat": 78, "lon": 32},
13.            {"lat": 82, "lon": 36},
14.            {"lat": 79, "lon": 38}
15.          ]
16.        }
17.      }
18.    }
19.  }
20. }
21. }
```

Здесь запрос ищет документы в пределах многоугольника, заданного набором точек. Значения широты и долготы точек можно изменить и задать свой многоугольник.

Точки в геополигональном запросе могут быть представлены в различных форматах:

- **Lat Long как массив** — значение `lat long` можно выразить как массив:

```

1. "location": {
2.   "points": [
3.     [-70, 40],
4.     [-80, 30],
5.     [-90, 20]
6.   ]
7. }
```

- **Lat Long как строка** — значение `lat long` можно выразить как строку:

```

1. "location": {
2.   "points": [
3.     "-70,40",
4.     "-80,30",
5.     "-90,20"
6.   ]
7. }
```

- **Lat Long как геохеш** — геохеш — это способ удобного представления местоположения в виде короткой буквенно-цифровой строки. См. следующее выражение:

```

1. "location": {
2.   "points": [
```

```

3.      "gbsuv7ztq",
4.      "-80,30",
5.      "-90,20"
6.    ]
7.  }

```

Можно использовать любое из этих представлений, чтобы определить многоугольник для геополигонального запроса.

Геополигональный запрос поможет найти документы в пределах определенной области, заданной многоугольником. Он полезен для поиска местоположений в заданных пользователем границах.

Границные запросы

Границные запросы можно использовать для поиска документов в пределах заданной границы. Вот пример такого запроса:

```

1. GET geo_index/_search
2. {
3.   "query": {
4.     "bool": {
5.       "must": {
6.         "match_all": {}
7.       },
8.       "filter": {
9.         "geo_bounding_box": {
10.           "location": {
11.             "top_left": {
12.               "lat": 43,
13.               "lon": -74
14.             },
15.             "bottom_right": {
16.               "lat": 42,
17.               "lon": -73
18.             }
19.           }
20.         }
21.       }
22.     }
23.   }
24. }

```

В этом примере запрос ищет документы с местоположением точки внутри указанного ограниченного поля. Значения широты и долготы можно изменить для левого верхнего и правого нижнего угла поля, чтобы определить область поиска.

Границы в подобном запросе могут быть заданы в разных форматах:

- **Lat Long как свойства:**

```

1. "location": {
2.   "top_left": {
3.     "lat": 40.73,
4.     "lon": -74.1
5.   },
6.   "bottom_right": {
7.     "lat": 40.01,
8.     "lon": -71.12
9.   }
10. }
```

- **Lat Long как массив:**

```

1. "location": {
2.   "top_left": [-74.1, 40.73],
3.   "bottom_right": [-71.12, 40.01]
4. }
```

- **Lat Long как строка:**

```

1. "location": {
2.   "top_left": "40.73,-74.1",
3.   "bottom_right": "40.01,-71.12"
4. }
```

- **Lat Long как геохеш:**

```

1. "location": {
2.   "top_left": "dr5r9ydj2y73",
3.   "bottom_right": "drj7teegpus6"
4. }
```

Для определения границ запроса можно использовать любое из этих представлений.

Границный запрос позволяет искать документы в пределах заданных границ. Он полезен для поиска местоположений в прямоугольной области.

Запросы геоформ

Запрос геоформы используется для поиска документов с маппингом геоформы, которая пересекается с заданной. Вот пример такого запроса с использованием формы конверта:

```

1. GET /geoshape_index/_search
2. {
```

```

3.   "query": {
4.     "bool": {
5.       "must": {
6.         "match_all": {}
7.       },
8.       "filter": {
9.         "geo_shape": {
10.           "location": {
11.             "shape": {
12.               "type": "envelope",
13.               "coordinates": [
14.                 [100, 1.0],
15.                 [102, 0.0]
16.               ]
17.             },
18.             "relation": "within".
19.           }
20.         }
21.       }
22.     }
23.   }
24. }
```

В этом примере запрос ищет документы, в которых форма местоположения пересекается с заданной формой конверта. Форма конверта задается левой верхней и правой нижней координатами в поле координат.

Координаты конверта можно изменить, чтобы задать свою форму. Параметр `type` задает тип фигуры, в данном случае `"envelope"`. Параметр `relation` задает пространственную связь между формой запроса и формой документа, где `"within"` означает, что форма запроса должна находиться внутри формы документа.

В запросах геоформ можно использовать другие типы фигур, например точки, линии, многоугольники, круги и т. д. Необходимо обновить параметр `type` в зависимости от используемой фигуры.

Запросы геоформ позволяют выполнять пространственный поиск на основе пересечения фигур. Это полезно для поиска документов, которые имеют определенную форму или находятся в пределах определенной формы.

ПРИМЕРЫ ИСПОЛЬЗОВАНИЯ

Геопоиск имеет множество применений, и невозможно разобрать их все. Многие из них в значительной степени используют геопоиск на основе расстояния. В таких сценариях мы ищем документы в определенном радиусе от исходной точки. Например, рассмотрим сценарий, в котором мы ищем рестораны

в радиусе 2 км от места, в котором мы находимся. Задача состоит в том, чтобы найти в индексе ресторанов Elasticsearch документы, в которых значения широты и долготы находятся в радиусе 2 км от нашего текущего местоположения. Примечательно, что это расстояние в 2 км является переменной, которую можно настраивать. Если мы не найдем подходящие рестораны в этом диапазоне, мы можем увеличить его до 5 км.

Другой пример — сервисы-агрегаторы такси. Эти приложения используют местоположение пользователя для отображения ближайших машин, доступных для заказа. Рассматривая только ближайшие такси, пользователи могут запросить поездку, не испытывая неудобств, связанных с большими расстояниями. Таким образом, местоположение пользователя является важнейшим параметром, так как оно формирует основу для всей сопутствующей функциональности.

Геопоиск открывает возможности поиска по близости в различных приложениях, позволяя пользователям находить ближайшие точки интереса, отслеживать объекты, рассчитывать расстояния до них и многое другое. Возможность настраивать радиус поиска в зависимости от конкретных требований расширяет набор подходящих сценариев использования геопоиска.

Поиск ресторана

Рассмотрим пример поиска ресторанов по местоположению с помощью Elasticsearch. Добавим несколько документов, включая местоположение пользователя и ближайшие рестораны. Для этого используем следующие координаты:

1. Местоположение пользователя: 28.580116, 77.051673
2. Ресторан 1: 28.594817, 77.046608
3. Ресторан 2: 28.593168, 77.041244
4. Ресторан 3: 28.601345, 77.026352
5. Ресторан 4: 28.590606, 77.062959

Сначала нужно создать индекс с именем "restaurants" и определить маппинг для поля "location" с типом "geo_point". Выполните следующий код:

```
1. PUT restaurants
2. {
3.   "mappings": {
4.     "properties": {
5.       "location": {
6.         "type": "geo_point"
7.       }
8.     }
9.   }
10. }
```

Итак, мы создали индекс ресторанов `restaurants` с маппингом типа геоточка. Далее добавим документы ресторанов с соответствующими координатами местоположения и названиями:

```
1. POST restaurants/_doc
2. {
3.   "name": "Ресторан 1",
4.   "location": "28.594817, 77.046608"
5. }
```

Здесь показано создание документа для первого ресторана, и оставшиеся четыре мы создадим аналогичным образом. Выполнив этот код, получим такой ответ:

```
1. {
2.   "_index" : "restaurants",
3.   "_type" : "_doc",
4.   "_id" : "8EvB-XEBL3C-ByPYIkGW",
5.   "_version" : 1,
6.   "result" : "created",
7.   "_shards" : {
8.     "total" : 2,
9.     "successful" : 1,
10.    "failed" : 0
11.  },
12.  "_seq_no" : 0,
13.  "_primary_term" : 1
14. }
```

После создания всех документов ресторана можем проверить их, выполнив следующую команду:

```
GET restaurants/_search
```

Эта команда выведет список всех документов в индексе "`restaurants`", это позволит убедиться, что все документы успешно добавлены.

Теперь выполним поиск ресторана на основе нашего текущего местоположения и посмотрим, как Elasticsearch отвечает на запросы о расстоянии. Сначала выполним поиск любых ресторанов в радиусе 1 км от нашего текущего местоположения (`28.580116, 77.051673`) с помощью такого кода:

```
1. GET restaurants/_search
2. {
3.   "query": {
4.     "bool": {
5.       "must": {
6.         "match_all": {}
7.       }
8.     }
9.   }
10. }
```

```
8.         "filter" : {
9.             "geo_distance" : {
10.                 "distance" : "1km",
11.                 "location" : "28.580116, 77.051673"
12.             }
13.         }
14.     }
15. }
16. }
```

Получим следующий ответ:

```
1. {
2.   "took" : 6,
3.   "timed_out" : false,
4.   "_shards" : {
5.     "total" : 1,
6.     "successful" : 1,
7.     "skipped" : 0,
8.     "failed" : 0
9.   },
10.  "hits" : {
11.    "total" : {
12.      "value" : 0,
13.      "relation" : "eq"
14.    },
15.    "max_score" : null,
16.    "hits" : [ ]
17.  }
18. }
```

Поскольку в радиусе 1 км мы не нашли ни одного ресторана, увеличим расстояние до 2 км и посмотрим, что выдаст Elasticsearch. Выполним код:

```
1. GET restaurants/_search
2. {
3.   "query": {
4.     "bool" : {
5.       "must" : {
6.         "match_all" : {}
7.       },
8.       "filter" : {
9.         "geo_distance" : {
10.           "distance" : "2km",
11.           "location" : "28.580116, 77.051673"
12.         }
13.       }
14.     }
15.   }
16. }
```

Получим следующий ответ:

```
1. }
2.   "took" : 2,
3.   "timed_out" : false,
4.   "_shards" : {
5.     "total" : 1,
6.     "successful" : 1,
7.     "skipped" : 0,
8.     "failed" : 0
9.   },
10.  "hits" : {
11.    "total" : {
12.      "value" : 3,
13.      "relation" : "eq"
14.    },
15.    "max_score" : 1.0,
16.    "hits" : [
17.      {
18.        "_index" : "restaurants",
19.        "_type" : "_doc",
20.        "_id" : "8EvB-XEBL3C-ByPYIkGW",
21.        "_score" : 1.0,
22.        "_source" : {
23.          "name" : "Ресторан 1",
24.          "location" : "28.594817, 77.046608"
25.        }
26.      },
27.      {
28.        "_index" : "restaurants",
29.        "_type" : "_doc",
30.        "_id" : "8UvE-XEBL3C-ByPYIUEs",
31.        "_score" : 1.0,
32.        "_source" : {
33.          "name" : "Ресторан 2",
34.          "location" : "28.593168, 77.041244"
35.        }
36.      },
37.      {
38.        "_index" : "restaurants",
39.        "type" : "doc",
40.        "_id" : "80vE-XEBL3C-ByPYkUG6",
41.        "_score" : 1.0,
42.        "_source" : {
43.          "name" : "Ресторан 4",
44.          "location" : "28.590606, 77.062959"
45.        }
46.      }
47.    ]
48.  }
49. }
```

В радиусе 2 км видим три ресторана: Ресторан 1, Ресторан 2 и Ресторан 4. Мы можем найти ближайшие рестораны, изменив параметр расстояния. Можно также задать расстояние в виде значения с плавающей точкой, например 1.8 км, чтобы отыскать рестораны в радиусе 1.8 км от текущего местоположения.

ГЕОАГРЕГАЦИЯ

В рассматриваемом сценарии можно выполнить агрегацию, чтобы сгруппировать рестораны на основе различных диапазонов расстояний. Агрегация позволяет предоставить пользователям опции фильтра, отображающие количество ресторанов в каждом диапазоне, например:

1. Рядом: 1 ресторан
2. В пределах 2 км: 2 ресторана
3. Более чем в 2 км: 1 ресторан

С помощью этих параметров фильтрации пользователи могут понять, сколько ресторанов доступно в пределах разных расстояний. Затем они могут выбрать любой вариант и просмотреть рестораны в выбранном диапазоне.

Примечание. Не волнуйтесь, если сейчас концепция агрегации показалась вам сложной. В следующей главе мы рассмотрим ее подробно.

Для агрегации на основе расстояния можно использовать следующий запрос:

```

1. POST restaurants/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "restaurant_in_range": {
6.       "geo_distance": {
7.         "field": "location",
8.         "unit": "km",
9.         "origin": {
10.           "lat": 28.580116,
11.           "lon": 77.051673
12.         },
13.         "ranges": [
14.           {
15.             "key": "Near",
16.             "from": 1,
17.             "to": 1.7
18.           },
19.           {
15.             "key": "Within 2 KM",
16.             "from": 1.7,

```

```

22.           "to": 2
23.       },
24.       {
25.           "key": "More than 2 KM away",
26.           "from": 2
27.       }
28.   ]
29. }
30. }
31. }
32. }
```

В этом запросе мы объединяем рестораны на основе их расстояния от местоположения пользователя ("lat": 28.580116, "lon": 77.051673). `size` установлен на 0, поскольку нам нужны только результаты агрегирования, а не сами документы.

Выполнив этот запрос, получим ответ:

```

1. {
2.   "took" : 1,
3.   "timed_out" : false,
4.   "_shards" : {
5.     "total" : 1,
6.     "successful" : 1,
7.     "skipped" : 0,
8.     "failed" : 0
9.   },
10.  "hits" : {
11.    "total" : {
12.      "value" : 4,
13.      "relation" : "eq"
14.    },
15.    "max_score" : null,
16.    "hits" : [ ]
17.  },
18.  "aggregations" : {
19.    "restaurant_in_range" : {
20.      "buckets" : [
21.        {
22.          "key": "Near",
23.          "from": 1,
24.          "to": 1.7
25.          "doc_count" : 1
26.        },
27.        {
28.          "key": "Within 2 km",
29.          "from": 1.7,
30.          "to": 2
31.          "doc_count" : 2
32.        },
33.      ],
34.      "summary" : {
35.        "min" : 1.0,
36.        "max" : 2.0,
37.        "count" : 4
38.      }
39.    }
40.  }
41.}
```

```
33.      {
34.          "key": "More than 2 km away",
35.          "from": 2,0,
36.          "doc_count" : 1
37.      }
38.  ]
39. }
40. }
41. }
```

В результате выполнения агрегации ответ будет включать в себя различные бакеты. У нас получилось три бакета: "Near" (Рядом) с одним документом, «Within 2 km» (В пределах 2 км) с двумя документами и "More than 2 km away" (Более 2 км) с одним документом. Эта агрегированная информация имеет большую ценность, поскольку дает пользователям представление о количестве ресторанов на каждом выбранном расстоянии, помогая им принять решение.

ЗАКЛЮЧЕНИЕ

В этой главе мы подробно рассмотрели работу с геоданными в Elasticsearch. Мы начали с изучения типов геоданных, в частности типов `geo-point` (геоточка) и `geo-shape` (геоформа). Для типа данных `geo-point` мы обсудили создание маппинга и хранение данных в индексе. Мы также рассмотрели различные способы представления геоданных.

Перейдя к типу данных `geo-shape`, мы изучили различные типы фигур, такие как `Point` (Точка), `LineString` (Линия), `Polygon` (Многоугольник), `MultiPoint` (Набор точек), `MultiLineString` (Набор линий), `Multi-Polygon` (Набор многоугольников), `Geometry Collection` (Коллекция геометрических объектов), `Envelope` (Конверт) и `Circle` (Окружность). Мы также узнали, как хранить данные в этих типах фигур.

Рассмотрев создание маппинга и документов, мы перешли к геозапросам. Мы обсудили запросы `geo_distance`, `geo_polygon`, `geo_bounding_box` и `geo_shape`. Эти запросы позволяют искать геопространственные данные на основе расстояния, многоугольников, заданных границ и конкретных фигур.

Кроме того, мы изучили сценарии использования геозапросов на примере данных о ресторанах. Мы узнали, как выполнять геопоиск и агрегацию, чтобы находить рестораны в пределах заданных расстояний и группировать их на основе диапазонов расстояний. Разобравшись с этими концепциями и примерами, вы получили знания, позволяющие эффективно работать с геоданными в Elasticsearch.

В следующей главе мы перейдем к теме агрегации данных. Мы изучим различные типы агрегации, включая агрегацию метрик, агрегацию бакетов, агрегацию пайплайнсов и агрегацию матриц.

ВОПРОСЫ

1. Расскажите о различных доступных типах геоданных.
2. Какие этапы включает в себя создание маппинга для данных типа геоточка?
3. Поясните процесс сохранения данных типа геоточка в Elasticsearch.
4. Раскройте концепцию данных типа геоформа и расскажите, как создать маппинг для них.
5. Подробно расскажите, что такое геозапросы и каких типов они бывают.

ГЛАВА 9

Анализ данных с помощью агрегаций Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели работу с геоданными в Elasticsearch, позволяющую использовать функции, основанные на местоположении, такие как поиск ближайших мест и поиск по координатам. Мы рассмотрели основные типы геоданных, а именно геоточку и геоформу, а также подробно изучили процессы маппинга и хранения геоданных в Elasticsearch. Кроме того, мы изучили различные доступные типы геоформ, включая `LineString`, `Polygon`, `MultiLineString`, `MultiPoint`, `MultiPolygon`, `Envelope` и `Circle`. Мы также получили представление о различных методах поиска геоданных, таких как запросы `geo_distance`, `geo_polygon`, `geo_bounding_box` и `geo_shape`. Наконец, мы проиллюстрировали важность геозапросов, применив их на практическом примере с ресторанами.

В этой главе мы сосредоточимся на агрегации в Elasticsearch. Агрегация позволяет группировать данные и выполнять статистические расчеты путем создания простых запросов. Используя агрегацию, можно проанализировать весь набор данных и получить их исчерпывающий обзор. Эта мощная функция позволяет формировать аналитические представления и извлекать полезные сведения из данных.

СТРУКТУРА

В этой главе:

- Введение в агрегацию Elasticsearch
- Агрегация метрик
- Агрегация бакетов
- Агрегация пайплайнов
- Агрегация матриц

ЦЕЛИ

После изучения этой главы у вас должно сформироваться понимание, как использовать агрегации Elasticsearch для анализа и обобщения данных. Мы рассмотрим различные типы агрегаций, включая агрегацию по метрикам, агрегацию по бакетам, агрегацию пайплайнов и агрегацию матриц. Агрегация поможет выполнить расширенный анализ данных, что сделает работу с Elasticsearch более эффективной.

ВВЕДЕНИЕ В АГРЕГАЦИЮ ELASTICSEARCH

В Elasticsearch существуют различные типы агрегации для разных сценариев использования. Основных типов агрегации четыре:

- **Агрегация бакетов** — предполагает группировку документов в группы — бакеты — на основе заданных критерии. Каждый бакет связан с ключом, который представляет собой определенный атрибут документа. Например, можно создать бакеты на основе поля "цена", такие как "менее 500", "от 500 до 1000", "от 1001 до 1500" или "более 1500". Документы относятся к соответствующему бакету на основе значения поля «цена». Аналогичным образом можно применять агрегацию бакетов к другим полям.
- **Агрегация метрик** — используется для вычисления метрик по набору документов. Она позволяет выводить метрики на основе определенных полей в документах. Например, можно вычислить среднее значение, сумму, минимум, максимум или любую другую статистическую метрику для конкретного поля.
- **Агрегация матриц** — работает с несколькими полями и создает матричную структуру на основе значений, содержащихся в документах. Она обеспечивает всесторонний обзор, представляя взаимосвязи между полями.

- **Агрегация пайплайнов** — предполагает объединение нескольких агрегаций для более сложного анализа. Выходные данные одной агрегации служат входными данными для другой, что позволяет выполнять расширенные вычисления и делать выводы.

Для построения агрегированных запросов в Elasticsearch необходимо понимать структуру агрегированного выражения. Вот она:

```
1. "aggregations|aggs" {  
2.     "<name of aggregation>": {  
3.         "<type of aggregation>": {  
4.             <aggregation body>  
5.         }  
6.     }  
7. }
```

- Первая строка служит началом запроса агрегации, в ней можно использовать выражение "`aggregations`" или краткую форму "`aggs`".
- Во второй строке указывается имя агрегации, чтобы идентифицировать ее в ответе.
- В третьей строке задается тип агрегации, например "`terms`" или другой доступный.
- Наконец, собственно тело агрегации, содержащее дополнительные параметры и специфичные конфигурации.

В последующих разделах мы более подробно рассмотрим каждый тип агрегации. Мы предоставим подробные объяснения и практические примеры, чтобы вы лучше поняли концепции и могли применить эти агрегации к своим наборам данных. Начнем с изучения агрегации бакетов.

АГРЕГАЦИЯ БАКЕТОВ

Агрегация бакетов — это мощная функция в Elasticsearch, которая позволяет создавать различные группы — бакеты — на основе заданных критериев. Бакеты можно определять с помощью уникальных значений полей или настраиваемых диапазонов. Полученная в результате агрегация представляет собой сnapshot набора данных с возможностью применения фильтров и получения представлений о распределении данных.

Например, рассмотрим сценарий, в котором существует набор данных о товарах с полем цены "`price`" (цена). Применив агрегацию по полю "`price`", мы можем создать такие ценовые группы, как "`less than $100`" (менее \$100), "`$100-$200`", "`$200-$300`" и т. д. Это позволит увидеть распределение товаров по ценовым категориям.

Чтобы объяснить механизм агрегации бакетов, воспользуемся набором данных "kibana_sample_data_flights", который можно бесплатно загрузить из интерфейса Kibana (рис. 9.1).

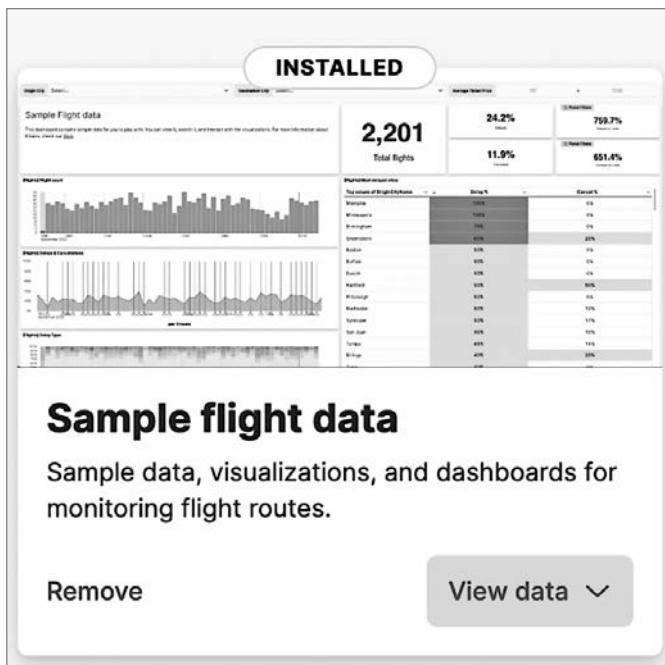


Рис. 9.1. Дашборд Sample flight data (Образец данных об авиарейсах) на Kibana

Индекс kibana_sample_data_flights имеет следующую структуру:

```

1. {
2.     "_index": "kibana_sample_data_flights",
3.     "_id": "9qS4IYkBkj0SG0U-XHC",
4.     "_score": 1,
5.     "_source": {
6.         "FlightNum": "9HY9SWR",
7.         "DestCountry": "AU",
8.         "OriginWeather": "Sunny",
9.         "OriginCityName": "Frankfurt am Main",
10.        "AvgTicketPrice": 841.2656419677076,
11.        "DistanceMiles": 10247.856675613455,
12.        "FlightDelay": false,
13.        "DestWeather": "Rain",
14.        "Dest": "Sydney Kingsford Smith International Airport",
15.        "FlightDelayType": "No Delay",
16.        "OriginCountry": "DE",

```

```

17.      "DayOfWeek": 0,
18.      "DistanceKilometers": 16492.32665375846,
19.      "timestamp": "2023-06-26T00:00:00",
20.      "DestLocation": {
21.          "lat": "-33.94609833",
22.          "lon": "151.177002"
23.      },
24.      "DestAirportID": "SYD",
25.      "Carrier": "Kibana Airlines",
26.      "Cancelled": false,
27.      "FlightTimeMin": 1030.7704158599038,
28.      "Origin": "Frankfurt am Main Airport",
29.      "OriginLocation": {
30.          "lat": "50.033333",
31.          "lon": "8.570556"
32.      },
33.      "DestRegion": "SE-BD",
34.      "OriginAirportID": "FRA",
35.      "OriginRegion": "DE-HE",
36.      "DestCityName": "Sydney",
37.      "FlightTimeHour": 17.179506930998397,
38.      "FlightDelayMin": 0
39.  }
40. }

```

Чтобы выполнить агрегацию по полю "DestCountry" (страна назначения) в индексе `kibana_sample_data_flights`, используем следующий запрос Elasticsearch:

```

1. GET kibana_sample_data_flights/_search
2. {
3.     "size": 0,
4.     "aggs": {
5.         "dest_country_aggregate": {
6.             "terms": {
7.                 "field": "DestCountry",
8.                 "size": 5
9.             }
10.        }
11.    }
12. }

```

Этот запрос извлекает результаты агрегации, не возвращая фактические документы. Агрегация выполняется по полю "DestCountry" с использованием типа "terms". Мы задаем параметр "size" равным 5, поскольку нас интересуют пять самых популярных стран назначения на основе количества документов. Результат агрегации будет выглядеть так:

```

1. "aggregations": {
2.     "dest_country_aggregate": {
3.         "doc_count_error_upper_bound": 0,
4.         "sum_other_doc_count": 5887,

```

```

5.     "buckets": [
6.         {
7.             "key": "IT",
8.             "doc_count": 2371
9.         },
10.        {
11.            "key": "US",
12.            "doc_count": 1987
13.        },
14.        {
15.            "key": "CN",
16.            "doc_count": 1096
17.        },
18.        {
19.            "key": "CA",
20.            "doc_count": 944
21.        },
22.        {
23.            "key": "JP",
24.            "doc_count": 774
25.        }
26.    ]
27. }
28. }
```

Вывод содержит информацию о странах назначения и количестве соответствующих им документов. В этом примере в топ-5 стран назначения входят "IT" (2371 рейс), "US" (1987 рейсов), "CN" (1096 рейсов), "CA" (944 рейса) и "JP" (774 рейса). Таким образом, агрегацию данных можно выполнить по любому индексу, используя любое поле. Существует несколько видов агрегации бакетов; рассмотрим их подробнее.

Агрегация диапазонов

Агрегация диапазонов позволяет пользователям задавать конкретные числовые диапазоны, на основе которых Elasticsearch создает отдельные бакеты с документами, соответствующими критериям. Значения 'to' и 'from' позволяют точно определить границы каждого бакета. В качестве примера обратимся к данным индекса `userdetails` из главы 7 «Возможности поиска»: освоение Query DSL запросов и техник поиска:

```

1. {
2.     "_index": "userdetails",
3.     "_type": "_doc",
4.     "_id": "9Et3CnIBL3C-ByPYDkF8",
5.     "_score": 1.0
6.     "_source": {
7.         "name": "Suresh",
8.         "gender": "male",
```

```

9.     "city": "Singapore",
10.    "age": 32
11.   }
12. }
```

Теперь используем эти данные индекса для создания агрегации диапазонов, чтобы сгруппировать пользователей по разным возрастным группам. Следующий пример демонстрирует, как работает агрегация диапазонов:

```

1. GET.userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_range": {
6.       "range": {
7.         "field": "age",
8.         "ranges": [
9.           {
10.             "key": "Age range 10-20 years1",
11.             "from": 10,
12.             "to": 20
13.           },
14.           {
15.             "key": "Age range 20-30 years",
16.             "from": 20,
17.             "to": 30
18.           },
19.           {
20.             "key": "Age range 30-40 years",
21.             "from": 30,
22.             "to": 40
23.           },
24.           {
25.             "key": "More than 40 years2",
26.             "from": 40
27.           }
28.         ]
29.       }
30.     }
31.   }
32. }
```

В приведенном выше запросе определены бакеты для разных значений возраста на основе поля `"age"`. Каждому бакету присваивается описательный ключ, чтобы усилить значение агрегации. Получив запрос, Elasticsearch выполняет агрегацию диапазонов и выдает следующий ответ:

¹ Возраст 10–20 лет. — Примеч. ред.

² Старше 40 лет. — Примеч. ред.

```

1. {
2.   "took": 0,
3.   "timed_out": false,
4.   "_shards": {
5.     "total": 1,
6.     "successful": 1,
7.     "skipped": 0,
8.     "failed": 0
9.   },
10.  "hits": {
11.    "total": {
12.      "value": 6,
13.      "relation": "eq"
14.    },
15.    "max_score": null,
16.    "hits": []
17.  },
18.  "aggregations": {
19.    "age_range": {
20.      "buckets": [
21.        {
22.          "key": "Age range 10-20 years",
23.          "from": 10.0,
24.          "to": 20.0,
25.          "doc_count": 1
26.        },
27.        {
28.          "key": "Age range 20-30 years",
29.          "from": 20.0,
30.          "to": 30.0,
31.          "doc_count": 1
32.        },
33.        {
34.          "key": "Age range 30-40 years",
35.          "from": 30.0,
36.          "to": 40.0,
37.          "doc_count": 2
38.        },
39.        {
40.          "key": "More than 40 years",
41.          "from": 40.0,
42.          "doc_count": 2
43.        }
44.      ]
45.    }
46.  }
47. }
```

Ответ содержит агрегацию диапазонов по полю "age" индекса userdetails. Результат включает различные бакеты, каждый из которых имеет ключевое значение, представляющее диапазон значений возраста, а в поле "doc_count"

отображается общее количество документов в бакетах. Эта эффективная функция позволяет выполнять агрегацию диапазонов для любых числовых полей в целях анализа данных.

Составная агрегация

Составная агрегация позволяет создавать бакеты, объединяющие несколько значений. Это особенно полезно для постраничного вывода больших наборов результатов агрегации. При использовании составной агрегации указывается параметр `sources`, который определяет источники, используемые для создания составного бакета. Для составных агрегаций доступны три типа источников, о которых рассказано ниже.

Термины

Если источником значений являются термины, порядок работы аналогичен агрегации терминов. Уникальные значения полей извлекаются для создания бакетов. Каждый бакет представляет собой уникальное значение поля вместе с количеством документов, имеющих это значение. Вот пример составной агрегации, при которой источником значений выступают термины:

```
1. GET.userdetails/_search
2. {
3.   "size": 1,
4.   "aggs": {
5.     "age_range": {
6.       "composite": {
7.         "sources": [
8.           {
9.             "age": {
10.               "terms": {
11.                 "field": "age"
12.               }
13.             }
14.           ]
15.         }
16.       }
17.     }
18.   }
19. }
```

Ответ будет содержать следующий результат:

```
1. "aggregations": {
2.   "age_range": {
3.     "after_key": {
4.       "age": 52
5.     },
6.     "buckets": [
```

```

7.      {
8.          "key": {
9.              "age": 15
10.         },
11.        "doc_count": 1
12.      },
13.      {
14.          "key": {
15.              "age": 20
16.         },
17.        "doc_count": 1
18.      },
19.      {
20.          "key": {
21.              "age": 32
22.         },
23.        "doc_count": 1
24.      },
25.      {
26.          "key": {
27.              "age": 37
28.         },
29.        "doc_count": 1
30.      },
31.      {
32.          "key": {
33.              "age": 42
34.         },
35.        "doc_count": 1
36.      },
37.      {
38.          "key": {
39.              "age": 52
40.         },
41.        "doc_count": 1
42.      }
43.    ]
44.  }
45. }
```

В ответе мы видим бакеты с ключами, представляющими значения возраста, и количество документов в поле `doc_count`.

Гистограмма

Гистограмма в качестве источника значений используется для создания бакетов с интервалами фиксированного размера с помощью числовых значений. Необходимо определить параметр `interval`, чтобы указать способ преобразования числовых значений поля в гистограммы. Рассмотрим следующий пример, в котором интервал возраста установлен равным 10:

```
1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_range": {
6.       "composite": {
7.         "sources": [
8.           {
9.             "age_histo": {
10.               "histogram": {
11.                 "field": "age",
12.                 "interval": 10
13.               }
14.             }
15.           }
16.         ]
17.       }
18.     }
19.   }
20. }
```

Ответ будет следующим:

```
1. "aggregations": {
2.   "age_range": {
3.     "after_key": {
4.       "age_histo": 50.0
5.     },
6.     "buckets": [
7.       {
8.         "key": {
9.           "age_histo": 10.0
10.         },
11.         "doc_count": 1
12.       },
13.       {
14.         "key": {
15.           "age_histo": 20.0
16.         },
17.         "doc_count": 1
18.       },
19.       {
20.         "key": {
21.           "age_histo": 30.0
22.         },
23.         "doc_count": 2
24.       },
25.       {
26.         "key": {
27.           "age_histo": 40.0
28.         },
29.         "doc_count": 1
30.       }
31.     }
32.   }
33. }
```

```

30.      },
31.      {
32.          "key": {
33.              "age_histo": 50.0
34.          },
35.          "doc_count": 1
36.      }
37.  ]
38. }
39. }
```

Таким образом можно выполнить составную агрегацию, используя гистограмму в качестве источника значений.

Гистограмма дат

Выбор гистограммы дат в качестве источника значений составной агрегации позволяет создавать бакеты на основе интервалов даты и времени. Принцип работы такой же, как для гистограммы, но вместо числовых значений для определения параметра интервала используются выражения даты/времени.

Используя гистограммы дат в качестве источника для составной агрегации, можно легко группировать и анализировать данные на основе определенных периодов. Это особенно полезно при работе с полями даты и при необходимости извлечь информацию из данных, основанных на времени. Вот пример составной агрегации с использованием гистограммы дат в качестве источника:

```

1. GET.userdetails/_search
2. {
3.     "size": 0,
4.     "aggs" : {
5.         "date_range": {
6.             "composite": {
7.                 "sources": [
8.                     {
9.                         "date_histo": {
10.                             "date_histogram": {
11.                                 "field": "timestamp",
12.                                 "interval": "month"
13.                             }
14.                         }
15.                     }
16.                 ]
17.             }
18.         }
19.     }
20. }
```

В приведенном выше примере составная агрегация выполняется по полю `timestamp` с гистограммой дат в качестве источника. Параметр `interval` имеет значение `"month"`, это означает, что данные будут сгруппированы в бакеты по месяцам. Ответ будет следующим:

```
1. "aggregations": {
2.   "date_range": {
3.     "after_key": {
4.       "date_histo": 1654051200000
5.     },
6.     "buckets": [
7.       {
8.         "key": {
9.           "date_histo": 1625097600000
10.          },
11.          "doc_count": 5
12.        },
13.        {
14.          "key": {
15.            "date_histo": 1627776000000
16.          },
17.          "doc_count": 3
18.        },
19.        {
20.          "key": {
21.            "date_histo": 1630454400000
22.          },
23.          "doc_count": 7
24.        }
25.      // Остальные бакеты...
26.    ]
27.  }
28. }
```

В ответе видим бакеты, созданные на основе гистограммы дат. Каждый бакет представляет собой определенный временной интервал (в данном случае — месяц) и включает количество документов (`doc_count`), попадающих в этот интервал.

Агрегация терминов

Агрегация терминов (term aggregation) в Elasticsearch позволяет создавать динамические бакеты на основе уникальных значений поля. Такая агрегация полезна, если необходимо получить общее представление о значениях полей, например, для понимания их распределения по документам или выявления самых частых и самых редких значений. Агрегация терминов позволяет получить эти данные. Вот пример:

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_terms_aggregation": {
6.       "terms": {
7.         "field": "age",
8.         "size": 10
9.       }
10.    }
11.  }
12. }
```

В запросе выше агрегация терминов применяется к полю `age` индекса `userdetails`. Ответ на этот запрос в Elasticsearch будет следующим:

```

1. "aggregations": {
2.   "age_terms_aggregation": {
3.     "doc_count_error_upper_bound": 0,
4.     "sum_other_doc_count": 0,
5.     "buckets": [
6.       {
7.         "key": 20,
8.         "doc_count": 2
9.       },
10.      {
11.        "key": 32,
12.        "doc_count": 2
13.      },
14.      {
15.        "key": 15,
16.        "doc_count": 1
17.      },
18.      {
19.        "key": 42,
20.        "doc_count": 1
21.      },
22.      {
23.        "key": 52,
24.        "doc_count": 1
25.      }
26.    ]
27.  }
28. }
```

Ответ показывает, что индекс `userdetails` содержит пять уникальных значений возраста: 15, 20, 32, 42 и 52. Каждое значение представлено в виде бакета, содержащего общее количество документов с этим конкретным значением возраста (`doc_count`). Кроме того, перед ключами бакетов в ответе содержатся такие ключи, как `doc_count_error_upper_bound` и `sum_other_doc_count`. Значение

`doc_count_error_upper_bound` представляет собой верхнюю границу ошибки, которая может возникнуть при подсчете документов для каждого значения поля. Значение `sum_other_doc_count` указывает на общее количество документов во всех бакетах. Если бакетов слишком много из-за большого количества уникальных значений, Elasticsearch выводит только верхние термины, а значение `sum_other_doc_count` содержит общее количество документов. Таким образом, чтобы провести анализ и получить подробные результаты, агрегацию терминов можно применять к любому полю.

Агрегация фильтра

Агрегация фильтра в Elasticsearch позволяет объединять данные в один бакет, применив к ним фильтр. Это полезно, если требуется сузить агрегацию до определенных условий фильтра, а не рассматривать весь набор данных. Разберем пример с данными индекса `userdetails` и определим средний возраст всех мужчин с помощью агрегации фильтра. Следующий запрос демонстрирует, как это работает:

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_filter": {
6.       "filter": {
7.         "term": {
8.           "gender": "male"
9.         }
10.      },
11.      "aggs": {
12.        "avg_age_male": {
13.          "avg": {
14.            "field": "age"
15.          }
16.        }
17.      }
18.    }
19.  }
20. }
```

В этом запросе мы сначала применяем фильтр для выбора только мужчин, задав условие для поля `gender` (пол) как `"male"` (мужской). Затем применяем агрегацию для вычисления среднего возраста с помощью параметра `avg`. После выполнения этого запроса Elasticsearch выдаст следующий ответ:

```

1. "aggregations": {
2.   "age_filter": {
3.     "doc_count": 5,
```

```

1.     "avg_age_male": {
2.         "value": 35.6
3.     }
4. }

```

Ответ содержит один бакет с ключом `avg_age_male`, определенным в запросе. Соответствующее значение равно 35.6, то есть средний возраст мужчин в этом наборе данных составляет 35.6. Используя агрегацию фильтра, можно применить условие фильтрации перед выполнением фактической агрегации, что позволяет получить целевые выводы на основе заданных критерииев.

Агрегация фильтров

Данная опция в Elasticsearch позволяет создавать агрегацию из нескольких бакетов, в которой каждый бакет создается на основе определенного фильтра. С помощью этой функции можно агрегировать индекс по нескольким фильтрам и создавать отдельные бакеты для каждого фильтра. Разберем агрегацию фильтров на примере с использованием индекса `userdetails`. В этом примере мы создадим два бакета: один для мужчин и один для женщин. См. следующий запрос:

```

1. GET userdetails/_search
2. {
3.     "size": 0,
4.     "aggs": {
5.         "messages": {
6.             "filters": {
7.                 "filters": {
8.                     "male": { "term": { "gender": "male" } },
9.                     "female": { "term": { "gender": "female" } }
10.                }
11.            }
12.        }
13.    }
14. }

```

Мы применили агрегацию фильтров по полю `gender` индекса `userdetails`. Мы задали два фильтра: один для мужчин и один для женщин. После выполнения этого запроса получим следующий ответ:

```

1. "aggregations": {
2.     "messages": {
3.         "buckets": {
4.             "female": {
5.                 "doc_count": 2
6.             },
7.             "male": {

```

```

8.           "doc_count": 5
9.       }
10.      }
11.    }
12. }
```

Ответ содержит два бакета: женщины и мужчины. В бакете женщин значение `doc_count` равно 2, оно указывает, что женщины встречаются в двух документах. Для бакета мужчин значение `doc_count` равно 5, оно указывает, что мужчины встречаются в пяти документах. Агрегация фильтров позволяет создавать несколько бакетов на основе разных фильтров, что дает возможность анализировать подмножества данных по отдельности. Если необходимо создать несколько бакетов на основе заданных условий фильтрации, агрегацию по фильтрам можно применить к любому полю.

Агрегация георасстояния

Агрегация георасстояния в Elasticsearch позволяет объединять документы на основе их расстояния от центральной точки. Это полезно, если требуется узнать количество документов на определенном расстоянии от заданной координаты. Чтобы показать, как работает агрегация георасстояния, рассмотрим пример с индексом ресторанов. В этом примере мы найдем количество ресторанов на определенном расстоянии от конкретной координаты.

Агрегация георасстояния выполняется с помощью следующего запроса:

```

1. POST restaurants/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "restaurant_in_range": {
6.       "geo_distance": {
7.         "field": "location",
8.         "unit": "km",
9.         "origin": {
10.           "lat": 28.580116,
11.           "lon": 77.051673
12.         },
13.         "ranges": [
14.           {
15.             "key": "Near",
16.             "from": 1,
17.             "to": 1.7
18.           },
19.           {
20.             "key": "Within 2 KM",
21.             "from": 1.7,
22.             "to": 2
23.           }
24.         ]
25.       }
26.     }
27.   }
28. }
```

```

23.      },
24.      {
25.          "key": "More than 2 KM away",
26.          "from": 2
27.      }
28.  ]
29. }
30. }
31. }
32. }
```

В этом примере мы выполняем поиск по индексу ресторана и применяем агрегацию по георасстоянию. Агрегация основана на поле местоположения, которое содержит геокоординаты каждого ресторана. В качестве единицы измерения расстояния мы указываем километры (**km**). Параметр `origin` задает центральную точку с широтой 28.580116 и долготой 77.051673, от которой отсчитываются расстояния. В параметре `ranges` мы определяем три диапазона расстояний: "Near" (Рядом) (от 1 до 1.7 км), "Within 2 km" (от 1.7 до 2 км) и "More than 2 km away" (Более 2 км).

После выполнения запроса Elasticsearch выдаст следующий ответ:

```

1. "aggregations": {
2.   "restaurant_in_range": {
3.     "buckets": [
4.       {
5.         "key": "Near",
6.         "from": 1.0,
7.         "to": 1.7,
8.         "doc_count": 3
9.       },
10.      {
11.        "key": "Within 2 km",
12.        "from": 1.7,
13.        "to": 2.0,
14.        "doc_count": 1
15.      },
16.      {
17.        "key": "More than 2 km away",
18.        "from": 2.0,
19.        "doc_count": 1
20.      }
21.    ]
22.  }
23. }
```

Ответ содержит три бакета, представляющих заданные значения расстояний. Бакет "Near" имеет значение `doc_count`, равное 3, оно указывает, что в пределах от 1 до 1.7 км от заданных координат находятся 3 ресторана. Бакет "Within 2 km"

имеет значение `doc_count` 1, указывающее, что в пределах от 1.7 до 2 км находится один ресторан. Бакет "More than 2 km away" имеет значение `doc_count`, равное 1, оно указывает, что на расстоянии более 2 км от заданных координат находится один ресторан.

Агрегация георасстояния полезна для анализа распределения данных по расстояниям. Он позволяет классифицировать документы, находящиеся на разных расстояниях от центральной точки.

АГРЕГАЦИЯ МЕТРИК

В Elasticsearch агрегация метрик используется для математических вычислений, таких как расчет суммы, среднего, минимума, максимума и т. д., применительно к значениям полей после агрегации документов. Существуют агрегации метрик с одним значением, которые возвращают одну метрику, и агрегации метрик с несколькими значениями, которые возвращают несколько метрик. Для лучшего понимания рассмотрим некоторые типы агрегации метрик с примерами.

Поиск минимума

Поиск минимума — это агрегация метрик с одним значением, которая возвращает минимальное значение числового поля после агрегации документов. Ее можно применить к полю непосредственно или с помощью скрипта. Вот пример с использованием индекса `userdetails`:

```
1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "min_age": {
6.       "min": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }
```

Ответ будет содержать минимальное значение возраста:

```
1. "aggregations": {
2.   "min_age": {
3.     "value": 15.0
4.   }
5. }
```

Видим, что минимальный возраст, полученный с помощью агрегации для поиска минимума, составляет 15 лет. Таким образом, данную агрегацию, входящую в набор агрегаций метрик, можно использовать в Elasticsearch для получения минимального числового значения поля индекса.

Поиск максимума

Поиск максимума — это агрегация метрик с одним значением, которая возвращает максимальное значение числового поля после агрегации документов. Ее можно применить к полю непосредственно или с помощью скрипта. Вот пример с использованием индекса `userdetails`:

```
1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "max_age": {
6.       "max": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }
```

Ответ будет содержать максимальное значение возраста:

```
1. "aggregations": {
2.   "max_age": {
3.     "value": 52.0
4.   }
5. }
```

Видим, что максимальный возраст, полученный с помощью агрегации для поиска максимума, составляет 52 года.

Вычисление среднего

Вычисление среднего — это агрегация метрик с одним значением, которая возвращает среднее значение числового поля после агрегации документов. Ее можно применить к полю непосредственно или с помощью скрипта. Вот пример с использованием индекса `userdetails`:

```
1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
```

```

5.      "avg_age": {
6.        "avg": {
7.          "field": "age"
8.        }
9.      }
10.    }
11.  }

```

Ответ будет содержать значение среднего возраста:

```

1. "aggregations": {
2.   "avg_age" : {
3.     "value": 30.428571428571427
4.   }
5. }

```

Здесь мы видим, что средний возраст, полученный с помощью агрегации для вычисления среднего, составляет 30.42 года.

Вычисление суммы

Вычисление суммы — это агрегация метрик с одним значением, которая возвращает сумму всех числовых значений числового поля после агрегации документов. Ее можно применить к полю непосредственно или с помощью скрипта. Вот пример с использованием индекса `userdetails`:

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "sum_age": {
6.       "sum": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }

```

Ответ будет содержать сумму всех возрастов:

```

1. "aggregations": {
2.   "sum_age": {
3.     "value": 213.0
4.   }
5. }

```

Здесь мы видим, что сумма всех возрастов, полученная с помощью агрегации суммы, составляет 213 лет.

Подсчет количества значений

Подсчет количества значений в Elasticsearch позволяет определить количество значений, извлеченных из агрегированных документов. Это особенно полезно, когда необходимо узнать частоту, с которой встречается значение, или количество значений определенного поля.

Ниже показан пример агрегации количества значений для поля "age" индекса "userdetails". Ответ на запрос будет содержать количество значений для поля "age".

Вот пример запроса:

```

1. GET.userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "total_age_fields_count": {
6.       "value_count": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }
```

- Параметр `size` имеет значение 0, это означает, что результаты поиска не должны возвращаться, поскольку нас интересуют только результаты агрегирования.
- В разделе `aggs` мы определяем агрегацию с именем "`total_age_fields_count`", используя агрегацию `value_count`.
- В параметре `field` задается поле, к которому требуется применить агрегацию количества значений, в данном случае это "`age`".

Ответ на этот запрос будет содержать раздел "`aggregations`", включающий агрегацию "`total_age_fields_count`". Поле "`value`" этой агрегации будет содержать количество значений поля "`age`", в данном примере равное 7.

```

1. "aggregations" : {
2.   "total_age_fields_count" : {
3.     "value" : 7
4.   }
5. }
```

Сбор статистики

Сбор статистики — это агрегация метрик с несколькими значениями, которая вычисляет статистику для значения числового поля путем агрегирования документов. Статистика включает такие показатели, как минимум, максимум,

сумма, количество, среднее и др. Значение можно взять из числового поля в документе или получить путем выполнения скрипта. Пример ниже демонстрирует использование агрегации статистики для поля "age" индекса "userdetails":

```
1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_stats": {
6.       "stats": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }
```

Этот запрос позволяет получить статистику для поля "age" из индекса "userdetails". Ответ будет следующим:

```
1. "aggregations": {
2.   "age_stats": {
3.     "count": 7,
4.     "min": 15.0,
5.     "max": 52.0,
6.     "avg": 30.428571428571427,
7.     "sum": 213.0
8.   }
9. }
```

Ответ, включающий агрегацию "age_stats", содержит такие параметры:

- **count** — количество документов с ненулевым значением для поля "age" (в данном примере 7 документов);
- **min** — минимальное значение поля "age" среди агрегированных документов (в данном примере 15.0);
- **max** — максимальное значение поля "age" (в данном примере 52.0);
- **avg** — среднее значение поля "age" (в данном примере 30.428571428571427);
- **sum** — сумма всех значений поля "age" (в данном примере 213.0).

Используя агрегацию статистики Elasticsearch, легко получить статистику для числового поля индекса. Она дает полезную информацию о распределении и сводную информацию о значениях поля.

Сбор расширенной статистики

Сбор расширенной статистики — это агрегация метрик с несколькими значениями, вычисляющая статистику для числового поля путем агрегирования документов.

Она позволяет получить дополнительные сведения, не входящие в базовую статистику, например значения таких параметров, как сумма квадратов, дисперсия, стандартное отклонение и границы стандартного отклонения. Значение можно взять из числового поля в документе или получить путем выполнения скрипта.

Пример ниже демонстрирует использование агрегации расширенной статистики для поля "age" индекса "userdetails":

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_extended_stats": {
6.       "extended_stats": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }
```

С помощью этого запроса мы получим расширенную статистику для поля "age" индекса "userdetails". Ответ будет таким:

```

1. "aggregations": {
2.   "age_extended_stats": {
3.     "count": 7,
4.     "min": 15.0,
5.     "max": 52.0,
6.     "avg": 30.428571428571427,
7.     "sum": 213.0,
8.     "sum_of_squares": 7541.0,
9.     "variance": 151.38775510204076,
10.    "std_deviation": 12.303973142933984,
11.    "std_deviation_bounds": {
12.      "upper": 55.0365177144394,
13.      "lower": 5.820625142703459
14.    }
15.  }
16. }
```

Ответ содержит агрегацию "age_extended_stats", включающую следующие сведения:

- **sum_of_squares** — сумма квадратов значений поля "age" (в данном примере 7541.0);
- **variance** — дисперсия значений поля "age" (в данном примере 151.38775510204076);
- **std_deviation** — стандартное отклонение значений поля "age" (в данном примере 12.303973142933984);

- `std_deviation_bounds` — верхняя и нижняя границы, в которые попадает большинство значений поля "age". Верхняя граница "upper" равна 55.0365177144394, а нижняя "lower" — 5.820625142703459. Используя агрегацию расширенной статистики, можно получить полные статистические данные для числового поля индекса в Elasticsearch. Она предоставляет полезную аналитическую информацию о распределении, изменчивости и границах значений полей.

Вычисление процентилей

Вычисление процентилей — это агрегация метрик с несколькими значениями, которая вычисляет один или несколько процентилей для значения числового поля путем агрегирования документов. Это позволяет получить представление о распределении и разбросе значений. Значение можно взять из числового поля в документе или получить путем выполнения скрипта. Пример ниже демонстрирует использование агрегации процентилей для поля "age" в индексе "userdetails":

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_percentile": {
6.       "percentiles": {
7.         "field": "age"
8.       }
9.     }
10.   }
11. }
```

С помощью этого запроса мы получим процентили для поля "age" индекса "userdetails". По умолчанию вычисляются следующие процентили: 1, 5, 25, 50, 75, 95 и 99. В ответе на запрос получим:

```

1. "aggregations": {
2.   "age_percentile": {
3.     "values": {
4.       "1.0": 15.0,
5.       "5.0": 15.0,
6.       "25.0": 20.0,
7.       "50.0": 32.0,
8.       "75.0": 39.5,
9.       "95.0": 52.0,
10.      "99.0": 52.0
11.    }
12.  }
13. }
```

Ответ содержит агрегацию "age_percentile" с рассчитанными значениями указанных процентиелей, в рассмотренном примере они следующие:

- 1.0 — значение 1-го процентиля равно 15.0, то есть 1 % значений поля "age" меньше или равно 15.0.
- 5.0 — значение 5-го процентиля равно 15.0, то есть 5 % значений поля "age" меньше или равно 15.0.
- 25.0 — значение 25-го процентиля равно 20.0, то есть 25 % значений поля "age" меньше или равно 20.0.
- 50.0 — значение 50-го процентиля равно 32.0 и представляет собой медиану, 50 % значений поля "age" меньше или равно 32.0.
- 75.0 — значение 75-го процентиля равно 39.5, то есть 75 % значений поля "age" меньше или равно 39.5.
- 95.0 — значение 95-го процентиля равно 52.0, то есть 95 % значений поля "age" меньше или равно 52.0.
- 99.0 — значение 99-го процентиля равно 52.0, то есть 99 % значений поля "age" меньше или равно 52.0.

Используя агрегацию процентиелей, можно проанализировать распределение и фактические процентили числового поля индекса в Elasticsearch. Эта информация помогает понять распределение и относительное положение значений в наборе данных.

АГРЕГАЦИЯ МАТРИЦ

Агрегация используется для создания матрицы путем извлечения значений из нескольких полей индекса. Она работает с заданными полями документов и не поддерживает скрипты.

Сбор матричной статистики

Сбор матричной статистики — это агрегация, работающая с одним или несколькими числовыми полями. Она вычисляет различные статистические показатели, такие как количество, среднее значение, дисперсия, асимметрия, эксцесс, ковариация, корреляция и т. д. Хотя этот метод можно применять к нескольким числовым полям, мы остановимся на поле "age" индекса "userdetails".

Пример ниже демонстрирует сбор матричной статистики для поля "age" индекса "userdetails":

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_matrix_stats": {
6.       "matrix_stats": {
7.         "fields": ["age"]},
8.       }
9.     }
10.   }
11. }
```

В этом примере мы получаем статистику по матрице для поля "age" индекса "userdetails". Ответ на представленный запрос будет таким:

```

1.   "aggregations": {
2.     "age_matrix_stats": {
3.       "doc_count": 7,
4.       "fields": [
5.         {
6.           "name": "age",
7.           "count": 7,
8.           "mean": 30.428571428571434,
9.           "variance": 176.61904761904768,
10.          "skewness": 0.4336291862091775,
11.          "kurtosis": 1.9621816119745739,
12.          "covariance": {
13.            "age": 176.61904761904768
14.          },
15.          "correlation": {
16.            "age": 1.0
17.          }
18.        }
19.      ]
20.    }
21. }
```

Ответ содержит набор статистических данных для поля "age". К ним относятся количество документов, содержащих поле "age", среднее значение поля "age", значения дисперсии, асимметрии и эксцесса, а также значения ковариации и корреляции для поля "age". Используя агрегацию статистики матрицы, можно узнать различные статистические показатели числового поля индекса в Elasticsearch. Они представляют ценность для анализа распределения и взаимосвязей данных.

АГРЕГАЦИЯ ПАЙПЛАЙНОВ

Агрегация пайплайнов — это разновидность агрегации в Elasticsearch, которая оперирует результатами других типов агрегации, что избавляет от необходимости анализировать документы. Существует два основных семейства агрегации пайплайнов: агрегация родительских (parent) и соседних (sibling) пайплайнов:

- **Parent.** В этом семействе агрегация выполняется для выходных данных родительской агрегации. Агрегации этого семейства могут генерировать новые бакеты или применять дополнительные агрегации к уже существующим. Предположим, у нас есть индекс заказов интернет-магазинов с такими полями, как "product_category" и "order_total". Нужно рассчитать среднюю сумму заказа для каждой категории товаров, а затем вычислить общую среднюю сумму заказа по всем категориям. Это можно сделать с помощью агрегации родительского пайплайна.

```

1. GET orders/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "category_avg": {
6.       "terms": {
7.         "field": "product_category"
8.       },
9.       "aggs": {
10.         "avg_total": {
11.           "avg": {
12.             "field": "order_total"
13.           }
14.         }
15.       }
16.     },
17.     "overall_avg": {
18.       "avg_bucket": {
19.         "buckets_path": "category_avg>avg_total".
20.       }
21.     }
22.   }
23. }
```

В этом примере агрегация "category_avg" вычисляет среднюю сумму заказа для каждой категории товаров. Затем агрегация "overall_avg" использует результаты "category_avg>avg_total" для вычисления среднего значения по всем категориям. Здесь агрегация родительского пайплайна ("overall_avg") оперирует результатами родительской агрегации ("category_avg").

- **Sibling.** Это семейство, в котором агрегация выполняется для выходных данных соседней агрегации. В данном случае агрегация пайплайна работает

на том же уровне, что и соседняя агрегация. Предположим, у нас есть индекс данных логов сайта с такими полями, как "timestamp" и "response_time". Нужно рассчитать максимальное время отклика для каждого часа дня, а затем общее максимальное время отклика по всем часам. Это можно сделать с помощью агрегации соседнего пайплайна.

```

1. GET website_logs/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "hourly_max": {
6.       "date_histogram": {
7.         "field": "timestamp",
8.         "calendar_interval": "hour"
9.       },
10.      "aggs": {
11.        "max_response_time": {
12.          "max": {
13.            "field": "response_time"
14.          }
15.        }
16.      }
17.    },
18.    "overall_max": {
19.      "max_bucket": {
20.        "buckets_path": "hourly_max>max_response_time".
21.      }
22.    }
23.  }
24. }
```

В этом примере агрегация "hourly_max" вычисляет максимальное время ответа для каждого часа дня, используя гистограмму дат. Затем агрегация "overall_max" использует результаты "hourly_max>max_response_time" для вычисления общего максимального времени ответа за все часы. Здесь агрегация соседнего пайплайна ("overall_max") оперирует результатами соседней агрегации ("hourly_max"). Эти примеры демонстрируют, как родительские и соседние агрегации могут использоваться для выполнения вычислений или агрегаций на выходе других агрегаций, что позволяет проводить более глубокий анализ и получать более подробные сведения из агрегированных данных.

Теперь подробнее рассмотрим различные типы агрегации пайплайнов и разберем практические примеры, чтобы лучше понять принципы их работы.

Вычисление среднего значения по бакетам

Агрегация среднего по бакетам (avg bucket aggregation) — это агрегация пайплайнов семейства соседних (sibling) пайплайнов. Она вычисляет среднее

значение результатов агрегации нескольких групп бакетов. Рассмотрим пример использования агрегации в Elasticsearch уже знакомого нам индекса `userdetails`:

```

1. GET.userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_range": {
6.       "range": {
7.         "field": "age",
8.         "ranges": [
9.           {
10.             "key": "Age range 10-20 years",
11.             "from": 10,
12.             "to": 20
13.           },
14.           {
15.             "key": "Age range 20-30 years",
16.             "from": 20,
17.             "to": 30
18.           },
19.           {
20.             "key": "Age range 30-40 years",
21.             "from": 30,
22.             "to": 40
23.           },
24.           {
25.             "key": "More than 40 years",
26.             "from": 40
27.           }
28.         ],
29.       },
30.       "aggs": {
31.         "avg_age": {
32.           "avg": {
33.             "field": "age"
34.           }
35.         }
36.       }
37.     },
38.     "total_avg_age": {
39.       "avg_bucket": {
40.         "buckets_path": "age_range>avg_age".
41.       }
42.     }
43.   }
44. }
```

В этом примере мы используем агрегацию диапазонов для группировки документов по возрастным группам. Внутри каждого диапазона вычисляем средний

возраст с помощью агрегации для вычисления среднего. Затем применяем агрегацию для вычисления среднего значения по бакетам, чтобы вычислить средний возраст по всем группам диапазона. Ответ будет содержать значение среднего возраста для каждого блока значений возраста, а также общий средний возраст по всем блокам:

```
1. "aggregations": {
2.   "age_range": {
3.     "buckets": [
4.       {
5.         "key": "Age range 10-20 years",
6.         "from": 10.0,
7.         "to": 20.0,
8.         "doc_count": 1,
9.         "avg_age": {
10.           "value": 15.0
11.         }
12.       },
13.       {
14.         "key": "Age range 20-30 years",
15.         "from": 20.0,
16.         "to": 30.0,
17.         "doc_count": 2,
18.         "avg_age": {
19.           "value": 20.0
20.         }
21.       },
22.       {
23.         "key": "Age range 30-40 years",
24.         "from": 30.0,
25.         "to": 40.0,
26.         "doc_count": 2,
27.         "avg_age": {
28.           "value": 32.0
29.         }
30.       },
31.       {
32.         "key": "More than 40 years",
33.         "from": 40.0,
34.         "doc_count": 2,
35.         "avg_age": {
36.           "value": 47.0
37.         }
38.       }
39.     ]
40.   },
41.   "total_avg_age": {
42.     "value": 28,5
43.   }
44. }
```

Ответ содержит среднее значение возраста для каждого бакета групп возрастов, а также параметр "total_avg_age", который представляет собой средний возраст по всем группам. Этот пример демонстрирует, как применить агрегацию для расчета средних значений из семейства соседних агрегаций, чтобы получить представление о среднем возрасте в разных группах возрастов и об общем среднем возрасте.

Вычисление максимального значения по бакетам

Вычисление максимального значения по бакетам — еще один тип агрегации семейства соседних пайплайнов. Он используется для получения максимального значения из выходных данных мультигрупповой соседней агрегации. Приведем пример использования такой агрегации в Elasticsearch для индекса userdetails:

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_range": {
6.       "range": {
7.         "field": "age",
8.         "ranges": [
9.           {
10.             "key": "Age range 10-30 years",
11.             "from": 10,
12.             "to": 30
13.           },
14.           {
15.             "key": "More than 30 years",
16.             "from": 30
17.           }
18.         ]
19.       },
20.       "aggs": {
21.         "avg_age": {
22.           "avg": {
23.             "field": "age"
24.           }
25.         }
26.       }
27.     },
28.     "total_max_age": {
29.       "max_bucket": {
30.         "buckets_path": "age_range>avg_age".
31.       }
32.     }
33.   }
34. }
```

В этом примере мы используем агрегацию диапазонов для группировки документов по возрастным группам. Внутри каждого диапазона мы вычисляем средний возраст с помощью агрегации для вычисления среднего. Затем мы применяем агрегацию для поиска максимального значения по бакетам, чтобы получить максимальный средний возраст по всем диапазонам.

Ответ будет содержать значение среднего возраста для каждой группы возрастов, а также максимальный средний возраст среди всех диапазонов:

```
1. "aggregations": {
2.   "age_range": {
3.     "buckets": [
4.       {
5.         "key": "Age range 10-30 years",
6.         "from": 10.0,
7.         "to": 30.0,
8.         "doc_count": 3,
9.         "avg_age": {
10.           "value": 18.33333333333332
11.         }
12.       },
13.       {
14.         "key": "More than 30 years",
15.         "from": 30.0,
16.         "doc_count": 4,
17.         "avg_age": {
18.           "value": 39.5
19.         }
20.       }
21.     ],
22.   },
23.   "total_max_age": {
24.     "value": 39.5,
25.     "keys": [
26.       "More than 30 years"
27.     ]
28.   }
29. }
```

Ответ содержит среднее значение возраста для каждой группы возрастов, а также параметр "**total_max_age**", который указывает максимальный средний возраст по всем группам. Поле "keys" показывает, какой диапазон способствовал достижению максимального среднего возраста.

Рассмотренный пример демонстрирует, как можно использовать агрегацию для поиска максимального значения из результатов соседней агрегации, чтобы получить информацию о наибольшем среднем возрасте в разных возрастных группах и общем максимальном среднем возрасте. Аналогичным образом можно использовать агрегацию для поиска минимума по бакетам,

которая получает минимальное значение вместо максимального с помощью опции "`min_bucket`".

Вычисление суммы значений по бакетам

Вычисление суммы значений по бакетам — это агрегация пайплайнов из семейства соседних, которая вычисляет сумму указанного поля из всех бакетов мультигрупповой соседней агрегации. Рассмотрим пример использования такой агрегации в Elasticsearch для знакомого нам индекса `userdetails`:

```

1. GET userdetails/_search
2. {
3.   "size": 0,
4.   "aggs": {
5.     "age_range": {
6.       "range": {
7.         "field": "age",
8.         "ranges": [
9.           {
10.             "key": "Age range 10-30 years",
11.             "from": 10,
12.             "to": 30
13.           },
14.           {
15.             "key": "More than 30 years",
16.             "from": 30
17.           }
18.         ]
19.       },
20.       "aggs": {
21.         "avg_age": {
22.           "avg": {
23.             "field": "age"
24.           }
25.         }
26.       }
27.     },
28.     "total_sum_age": {
29.       "sum_bucket": {
30.         "buckets_path": "age_range>avg_age".
31.       }
32.     }
33.   }
34. }
```

В этом примере мы используем агрегацию диапазонов для группировки документов по возрастным группам. Внутри каждого диапазона мы вычисляем средний возраст с помощью агрегации для вычисления среднего. Затем

применяем агрегацию для вычисления суммы значений по бакетам, чтобы вычислить сумму средних значений возраста из всех групп диапазона.

Ответ содержит значение среднего возраста для каждой группы возрастов, а также общую сумму средних возрастов:

```

1. "aggregations": {
2.   "age_range": {
3.     "buckets": [
4.       {
5.         "key": "Age range 10-30 years",
6.         "from": 10.0,
7.         "to": 30.0,
8.         "doc_count": 3,
9.         "avg_age": {
10.           "value": 18.3333333333332
11.         }
12.       },
13.       {
14.         "key": "More than 30 years",
15.         "from": 30.0,
16.         "doc_count": 4,
17.         "avg_age": {
18.           "value": 39.5
19.         }
20.       }
21.     ],
22.   },
23.   "total_sum_age": {
24.     "value": 57.8333333333333
25.   }
26. }
```

Видим, что каждый бакет ответа содержит поле `avg_age`, а внутри него — поле `doc_count`, которое показывает средний возраст для этого бакета. Мы применили дополнительную агрегацию, чтобы получить сумму полей `avg_age` по всем бакетам.

ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели агрегацию в Elasticsearch и узнали, как с ее помощью эффективно собирать и обобщать данные. Мы начали с изучения основ агрегации, понимания ее цели и значения для анализа данных. Затем мы исследовали различные типы агрегации, включая агрегацию бакетов, агрегацию метрик, агрегацию матриц и агрегацию пайплайнов. Для лучшего понимания мы рассмотрели практические примеры, иллюстрирующие теоретические

концепции в действии. К концу изучения этой главы вы должны иметь навыки использования агрегаций в Elasticsearch, чтобы извлекать полезные сведения из имеющихся данных.

В следующей главе мы перейдем к оптимизации производительности Elasticsearch. Мы вооружимся знаниями о том, как точно настроить скорость индексирования, скорость поиска и использование диска. Кроме того, мы рассмотрим лучшие практики, повышающие эффективность и производительность кластера Elasticsearch.

Приготовьтесь раскрыть весь потенциал Elasticsearch, поскольку в следующей главе мы займемся повышением производительности!

ВОПРОСЫ

1. Что означает агрегация Elasticsearch?
2. Какие существуют типы агрегации в Elasticsearch?
3. Приведите пример, иллюстрирующий использование агрегации диапазонов.
4. Как определить агрегацию георасстояний в Elasticsearch?
5. Кратко объясните принцип работы агрегации матриц в Elasticsearch.

ГЛАВА 10

Настройка производительности

ВВЕДЕНИЕ

В предыдущей главе мы изучили мощную концепцию агрегации Elasticsearch и разобрались в том, как эффективно обращаться с данными и анализировать их. Мы рассмотрели различные типы агрегации, включая агрегацию по бакетам, агрегацию метрик, агрегацию матриц и агрегацию пайплайнов. Эти агрегации представляют собой ценные инструменты для извлечения важной информации из данных Elasticsearch.

В этой главе мы сместим акцент на повышение производительности Elasticsearch. Мы рассмотрим, почему важно оптимизировать такие ключевые параметры, как скорость индексирования, скорость поиска и использование диска в экосистеме Elasticsearch. Тонкая настройка этих параметров производительности позволит раскрыть весь потенциал Elasticsearch и обеспечить эффективное управление данными.

Кроме того, мы подробно рассмотрим лучшие практики для Elasticsearch. Эти практики позволяют использовать возможности платформы в полной мере. Выполняя эти рекомендации, мы сможем повысить надежность, масштабируемость и общую производительность среды Elasticsearch.

СТРУКТУРА

В этой главе:

- Стратегии оптимизации производительности Elasticsearch
- Оптимизация Elasticsearch для работы с большими объемами данных

- Настройка скорости индексирования Elasticsearch
- Настройка скорости поиска в Elasticsearch
- Настройка Elasticsearch для оптимизации использования диска
- Лучшие практики Elasticsearch

ЦЕЛИ

После изучения этой главы вы сможете повышать производительность Elasticsearch и использовать лучшие практики для оптимизации функциональности платформы. Это позволит увеличить эффективность и отзывчивость кластера Elasticsearch, обеспечивая его оптимальную работу для различных сценариев использования.

СТРАТЕГИИ ОПТИМИЗАЦИИ ПРОИЗВОДИТЕЛЬНОСТИ ELASTICSEARCH

Elasticsearch славится своими возможностями быстрого индексирования, поиска и агрегации данных. Однако с увеличением объема данных и изменением условий использования производительность Elasticsearch может снизиться. Различные приложения предъявляют разные требования: одни более требовательны к индексированию, для других на первом плане стоят возможности поиска. Достижение идеального баланса становится сложной задачей, требующей компромиссов, основанных на конкретных сценариях использования. В этой главе мы рассмотрим различные методы оптимизации Elasticsearch, включая настройку скорости индексирования, скорости поиска, использования диска, а также применение лучших практик.

Пытаться оптимизировать все параметры одновременно нецелесообразно. Важно понять приоритеты бизнеса и найти компромиссные решения, сосредоточившись на наиболее важных областях. В оценке эффективности мер оптимизации и их влияния на общую производительность решающую роль играет бенчмаркинг. Эффективнее всего проводить оптимизацию параметров по одному, оценивая ее влияние, прежде чем двигаться дальше.

Если разрабатываемое приложение является преимущественно индексируемым, основное внимание стоит уделить индексированию данных, а не поисковым операциям. В таких случаях целесообразно изучить варианты настройки скорости индексирования. И наоборот, если приложение в значительной степени зависит от функциональности поиска, целесообразно использовать методы повышения скорости поиска.

ОПТИМИЗАЦИЯ ELASTICSEARCH ДЛЯ РАБОТЫ С БОЛЬШИМИ ОБЪЕМАМИ ДАННЫХ

Рассмотрим стратегии оптимизации производительности Elasticsearch при работе с большими объемами данных. По мере роста объемов данных стандартная конфигурация Elasticsearch может оказаться недостаточной для поддержания оптимальной производительности. Применение специальных методов и лучших практик гарантирует стабильно быструю и эффективную индексацию, поиск и агрегирование данных в Elasticsearch.

Прежде всего необходимо оценить процесс индексирования. Массовое индексирование обычно более эффективно, чем индексирование отдельных документов, так как оно снижает накладные расходы на прием-передачу по сети. Кроме того, использование Elasticsearch Bulk API позволяет отправлять несколько документов в одном запросе, что еще больше увеличивает скорость индексирования. Размер пакета индексирования можно регулировать, чтобы найти оптимальный баланс между производительностью и потреблением ресурсов.

Поговорим теперь об аппаратной инфраструктуре. Для эффективной работы Elasticsearch важно иметь достаточное количество ресурсов, особенно оперативной памяти и хранилища. Выделение соответствующего размера кучи для **виртуальной машины Java (JVM)** Elasticsearch имеет решающее значение в деле эффективного индексирования и поиска. Кроме того, использование высокопроизводительных дисков, таких как **твердотельные накопители (SSD)**, значительно повышает общую производительность системы.

Существуют различные методы повышения производительности поиска. Кэширование результатов поиска как на уровне Elasticsearch, так и на уровне приложений ускоряет выполнение последующих поисковых запросов. Также можно использовать фильтры запросов и кэшировать их для повторного использования. Кроме того, для оптимизации операций поиска в Elasticsearch важна тщательная разработка модели данных и маппинга, что обеспечит надлежащее индексирование и анализ соответствующих полей.

Еще один важный аспект, который следует учитывать, — это конфигурация шардов. Распределение данных по нескольким шардам позволяет распараллелить операции индексирования и поиска, что повышает производительность. Однако чтобы избежать чрезмерных накладных расходов и нехватки ресурсов, необходимо найти баланс между количеством шардов и аппаратными ресурсами.

И наконец, крайне важен мониторинг производительности Elasticsearch. Использование таких инструментов, как Elasticsearch Monitoring API, и внешних

решений для мониторинга позволяет отслеживать метрики системы, выявлять узкие места и проводить обоснованную оптимизацию. Регулярный пересмотр и тонкая настройка конфигурации кластера на основе показателей производительности обеспечивают оптимальную производительность по мере роста объемов данных.

Реализуя эти стратегии и контролируя производительность Elasticsearch на постоянной основе, можно оптимизировать возможности этого инструмента для обработки больших объемов данных, обеспечивая быстрое и эффективное индексирование, поиск и агрегацию.

НАСТРОЙКА СКОРОСТИ ИНДЕКСИРОВАНИЯ ELASTICSEARCH

Рассмотрим имеющиеся варианты оптимизации скорости индексирования в Elasticsearch. Прежде чем приступить к оптимизации, важно понять, является ли приложение ориентированным на поиск или на индексирование. Это поможет принимать решения по оптимизации, исходя из конкретных требований сценария использования. В некоторых сценариях может возникнуть потребность в одновременном получении большого количества данных, например, в процессе миграции данных. В таких случаях можно временно оптимизировать производительность индекса и отменить изменения после завершения миграции. Такая гибкость позволяет регулировать производительность кластера Elasticsearch в соответствии с требованиями приложения.

Массовые запросы вместо одного

Чтобы повысить производительность индексирования в Elasticsearch, можно использовать массовые запросы вместо отправки отдельных запросов на индексирование каждого документа. Однако чтобы определить оптимальный размер данных для включения в один массовый запрос, требуется провести сравнительный анализ и эксперименты. Например, если необходимо проиндексировать большое количество документов, можно начать с пакета среднего размера, например 100–200 документов, и постепенно увеличивать его, пока не возникнут проблемы производительности. Такой подход позволит найти оптимальное количество документов, для которых можно эффективно реализовать массовое индексирование с учетом возможностей кластера Elasticsearch. Здесь важно найти баланс, поскольку слишком большое количество документов в одном массовом запросе может создать нагрузку на память кластера Elasticsearch.

Чтобы выполнить массовый запрос на индексирование, можно воспользоваться следующей командой:

```
1. Post _bulk
2. { "index" : { "_index" : "users", "_id" : "1" } }
3. { "name" : "Sophia Julian" }
4. { "index" : { "_index" : "users", "_id" : "2" } }
5. { "имя" : "John" }
6. { "create" : { "_index" : "users", "_id" : "3" } }
7. { "name" : "Sam Kuran" }
8. { "update" : { "_id" : "2", "_index" : "users" } }
9. { "doc" : { "name" : "Sam Paul" } }
```

В этом примере в рамках одного массового запроса выполняется несколько операций. Мы создаем документы с идентификаторами 1, 2 и 3 и обновляем документ с идентификатором 2. Кроме того, массовый запрос поддерживает операции удаления.

Ответ на этот массовый запрос будет содержать результат всех операций. Вот пример ответа:

```
1. {
2.   "took" : 384,
3.   "errors" : false,
4.   "items" : [
5.     {
6.       "index" : {
7.         "_index" : "users",
8.         "_type" : "_doc",
9.         "_id" : "1",
10.        "_version" : 1,
11.        "result" : "created",
12.        "_shards" : {
13.          "total" : 2,
14.          "successful" : 1,
15.          "failed" : 0
16.        },
17.        "_seq_no" : 0,
18.        "_primary_term" : 1,
19.        "status" : 201
20.      }
21.    },
22.    {
23.      "index" : {
24.        "_index" : "users",
25.        "_type" : "_doc"
26.        "_id" : "2",
27.        "_version" : 1,
28.        "result" : "created",
29.        "_shards" : {
```

```
30.          "total" : 2,
31.          "successful" : 1,
32.          "failed" : 0
33.        },
34.        "_seq_no" : 1,
35.        "_primary_term" : 1,
36.        "status" : 201
37.      }
38.    },
39.  }
40.  "create" : {
41.    "_index" : "users",
42.    "_type" : "_doc",
43.    "_id" : "3",
44.    "_version" : 1,
45.    "result" : "created"
46.    "_shards" : {
47.      "total" : 2,
48.      "successful" : 1,
49.      "failed" : 0
50.    },
51.    "_seq_no" : 2,
52.    "_primary_term" : 1,
53.    "status" : 201
54.  },
55.  },
56.  {
57.    "update" : {
58.      "_index" : "users",
59.      "_type" : "_doc",
60.      "_id" : "2",
61.      "_version" : 2,
62.      "result" : "updated",
63.      "_shards" : {
64.        "total" : 2,
65.        "successful" : 1,
66.        "failed" : 0
67.      },
68.      "_seq_no" : 3,
69.      "_primary_term" : 1,
70.      "status" : 200
71.    }
72.  }
73. ]
74. }
```

Ответ содержит информацию по каждой операции в рамках массового запроса, такой как создание, обновление или удаление документа. Эта информация поможет отследить состояние каждой операции и обеспечить успешное выполнение массового запроса.

Использование массовых запросов позволяет значительно повысить производительность индексирования в Elasticsearch, поскольку сокращает количество обходов сети и оптимизирует эффективность процесса индексирования.

Разумное использование кластера Elasticsearch

Чтобы оптимизировать производительность кластера Elasticsearch, важно грамотно распределять его ресурсы. Один из подходов заключается в использовании нескольких потоков и процессов для обработки массовых запросов к большим наборам данных. Это позволит максимально эффективно использовать возможности кластера для обработки большого количества запросов. Чтобы задействовать все имеющиеся ресурсы кластера Elasticsearch, используйте несколько потоков или процессов. Однако чтобы определить оптимальное количество исполнимых модулей или потоков, необходимо провести сравнительный анализ, поскольку в разных кластерах оно может отличаться.

Чтобы определить количество исполнимых модулей, можно постепенно увеличивать их число, отслеживая показатели использования ввода-вывода и процессора в кластере. По достижении точки насыщения ввода-вывода или процессора определяется оптимальное количество исполнимых модулей для текущей конфигурации кластера. Этот процесс позволяет точно настроить кластер для достижения оптимальной производительности, не перегружая его.

Комбинируя массовое индексирование и оптимизацию количества исполнимых модулей, можно значительно повысить производительность индексирования данных в кластере Elasticsearch. Такой подход позволяет использовать все доступные ресурсы и обеспечить эффективное индексирование.

Увеличение интервала обновления

От интервала обновления в Elasticsearch зависит, как часто проиндексированные документы становятся доступными для поисковых операций. Интервал обновления по умолчанию составляет 1 секунду, это означает, что новые проиндексированные документы могут быть найдены максимум через 1 секунду. Однако обновление происходит только в том случае, если индексы Elasticsearch получили один или несколько поисковых запросов в течение последних 30 секунд. Хотя интервал обновления в 1 секунду обеспечивает возможность поиска в реальном времени, он может негативно повлиять на производительность индексирования из-за связанных с этим накладных расходов.

Для оптимизации производительности индексирования рекомендуется увеличить интервал обновления (`refresh_interval`). Этого можно добиться, изменив параметр `index.refresh_interval` либо в файле конфигурации Elasticsearch, либо на каждом уровне индекса с помощью запроса. Для сценариев, предусматривающих массовое индексирование, рекомендуется установить более высокий интервал, например 30 секунд или более, чтобы избежать частого обновления индекса каждую секунду. Увеличение интервала обновления позволяет значительно повысить производительность индексирования.

Отключение репликации

Отключение репликации индексов может стать эффективной стратегией повышения производительности индексирования, особенно при загрузке большого объема данных. Установив значение `index.number_of_replicas` равным 0, можно оптимизировать производительность индексирования. Этот параметр можно настроить либо в файле `elasticsearch.yml`, либо через API настроек. Однако важно отметить, что отключение репликации создает риск потери данных в случае отказа узла. Чтобы снизить этот риск, рекомендуется обеспечить избыточность данных, сохранив их в другом месте. После успешной записи данных репликацию можно снова включить, изменив параметр `index.number_of_replicas`. Отключение репликации позволит избежать ненужных накладных расходов, связанных с репликацией данных на нескольких узлах, и тем самым повысить производительность индексирования в процессе первоначальной загрузки данных.

Использование автоматически генерируемых идентификаторов

Во время индексирования документов Elasticsearch несет дополнительные накладные расходы на обработку при использовании предварительно заданных значений поля ID индекса. Это связано с тем, что Elasticsearch должен проверить уникальность идентификатора и убедиться в его наличии в существующих документах. Если идентификатор уже присутствует в шарде, Elasticsearch вернет ошибку. Чтобы минимизировать эти накладные расходы и повысить производительность индексирования, рекомендуется использовать автоматически генерируемые идентификаторы. Это позволяет избежать необходимости проверки идентификаторов, что повышает производительность индексирования. Поэтому если особые требования к использованию пользовательских идентификаторов отсутствуют, рекомендуется для повышения производительности использовать автогенерируемые идентификаторы.

Настройка размера буфера индексирования

В сценариях с интенсивным индексированием важно убедиться, что размер буфера индексирования оптимизирован для эффективного индексирования данных. По умолчанию для индексирования Elasticsearch выделяет 512 Мбайт на шард. Чтобы обеспечить бесперебойное индексирование данных при больших нагрузках, очень важно правильно настроить параметр `indices.memory.index_buffer_size`. Его можно задать как процент от размера кучи Java или как абсолютное значение в байтах. В конфигурации по умолчанию для размера буфера индексирования отводится 10 % памяти JVM, чего обычно достаточно для многих сценариев использования. Например, для JVM объемом 10 Гбайт размер буфера должен составлять 1 Гбайт, что обеспечивает достаточно места для размещения двух шардов с сильным индексированием. Определить оптимальный размер буфера можно с помощью бенчмаркинга, который позволяет точно настроить производительность в зависимости от конкретных требований и характеристик рабочей нагрузки.

Использование более быстрого оборудования

Для дальнейшего повышения производительности индексирования стоит использовать более быстрые аппаратные компоненты, такие как твердотельные накопители. Они обеспечивают более высокую производительность по сравнению с традиционными вращающимися дисками. Рекомендуется выбирать локальное хранилище, а не удаленные файловые системы, такие как **Server Message Block (SMB)** или **Network File System (NFS)**, а также избегать виртуализированных решений хранения, таких как AWS Elastic Block Storage. Хотя облачное хранилище обеспечивает удобство и приемлемую производительность, выделенное локальное хранилище превосходит его по скорости текущей обработки.

Выбор в пользу SSD-накопителей и локальных хранилищ может значительно повысить производительность индексирования.

Выделение памяти для кэша файловой системы

Использование кэша файловой системы позволяет повысить производительность за счет буферизации операций ввода-вывода. Кэш служит в качестве буфера, повышая скорость получения данных. Для достижения оптимальных результатов рекомендуется выделять под файловый кэш примерно половину памяти узла, на котором запущен Elasticsearch. Эффективное использование кэша файловой системы позволяет значительно повысить производительность операций поиска.

НАСТРОЙКА СКОРОСТИ ПОИСКА В ELASTICSEARCH

Рассмотрев различные подходы к оптимизации производительности индексирования, изучим способы повышения скорости поиска, особенно в сценариях, где приложение ориентировано на поиск, а не на индексирование. В некоторых приложениях индексирование происходит редко, а операции поиска выполняются значительно чаще. Например, в приложении интернет-магазина сведения о товаре индексируются однократно, а затем их ищут множество пользователей. В таких случаях основной задачей является повышение производительности поиска в кластере Elasticsearch. Чтобы удовлетворить специфические требования приложения, производительность кластера Elasticsearch можно соответствующим образом корректировать. В следующих разделах мы рассмотрим способы оптимизации производительности поиска Elasticsearch.

Моделирование документов

Моделирование документов — важнейшее условие оптимизации производительности поиска в Elasticsearch. При разработке структуры документа важно согласовать ее с требованиями приложения. Один из ключевых моментов — избегать соединений или отношений «родитель — ребенок» в модели данных. Хранение всех необходимых атрибутов в одном индексе, например данных о товаре на странице с подробной информацией о товаре, обеспечивает быстрое время загрузки. В противном случае возникают небольшие задержки при получении данных из разных индексов. Моделирование документов играет важную роль в настройке производительности поиска Elasticsearch.

Поиск по меньшему количеству полей

Для повышения производительности поиска рекомендуется ограничить количество полей, по которым ведется поиск. Добавление дополнительных полей в операцию поиска с помощью `query_string` или `multi_search` увеличивает время, необходимое для получения результатов. Полезно по возможности сокращать число полей, по которым проводится поиск. Однако в некоторых сценариях требуется поиск по нескольким полям. В таких случаях Elasticsearch предлагает решение, позволяющее копировать значения нескольких полей в одно поле. Для этого используется директива `copy_to`.

Рассмотрим пример с двумя полями: `first_name` и `last_name`. Чтобы применить поисковый запрос к обоим полям, используем директиву `copy_to` для создания копии значений полей в другом поле, например `name`. Вот пример маппинга:

```
1. PUT user_details
2. {
```

```

3.   "mappings": {
4.     "properties": {
5.       "name": {
6.         "type": "text"
7.       },
8.       "first_name": {
9.         "type": "text",
10.        "copy_to": "name"
11.      },
12.      "last_name": {
13.        "type": "text",
14.        "copy_to": "name"
15.      }
16.    }
17.  }
18. }
```

В этом маппинге оба поля, `first_name` и `last_name`, копируются в поле `name`. Это позволяет выполнять поиск данных по полю `name`, даже если пользователь вводит данные в поля `first_name` или `last_name`. Чтобы проверить, что функциональность работает, добавим документ и выполним поисковый запрос по новому полю:

```

1. POST user_details/_doc/
2. {
3.   "first_name" : "Sophia",
4.   "last_name" : "Julian"
5. }
```

С помощью команды выше мы добавили один документ в созданный индекс `user_details`. Теперь выполним поиск по записи, чтобы убедиться, что данные скопированы в новое поле, в котором мы проведем поиск. Выполним следующую команду для поиска по новому полю:

```
GET user_details/_search?q=name:Sophia
```

В примере видно, что мы выполняем URI-поиск, где имя поля — это `name`, но мы не предоставили это поле для создания документа. Теперь посмотрим, какой ответ выдает Elasticsearch после выполнения команды выше:

```

1. {
2.   "took" : 197,
3.   "timed_out" : false,
4.   "_shards" : {
5.     "total" : 1,
6.     "successful" : 1,
7.     "skipped" : 0,
8.     "failed" : 0
9.   },
10.  "hits" : {
```

```

11.     "total" : {
12.         "value" : 1,
13.         "relation" : "eq"
14.     },
15.     "max_score" : 0.6931471,
16.     "hits" : [
17.         {
18.             "_index" : "user_details",
19.             "_type" : "_doc",
20.             "_id" : "W_KNLnIBV6DMUX9AYs9b",
21.             "_score" : 0.6931471,
22.             "_source" : {
23.                 "first_name" : "Sophia",
24.                 "last_name" : "Julian"
25.             }
26.         }
27.     ]
28. }
29. }
```

Итак, мы рассмотрели, как использовать поле `name` для поиска записей. Это полезно в случаях, когда целесообразно переместить данные из нескольких полей в одно, чтобы сократить количество полей для поиска данных и тем самым улучшить производительность поиска в Elasticsearch.

Предварительное индексирование данных

Одним из способов оптимизации индексирования данных является использование информации о шаблонах поиска. Добавление соответствующих данных в процессе индексирования позволяет повысить производительность поиска. При агрегации данных для вывода диапазонов (например, размеров обуви) выполнение агрегации во время поиска может быть дорогостоящим. Чтобы избежать этой проблемы, более эффективно — указывать данные о диапазоне в процессе индексирования, тем самым сокращая накладные расходы агрегации и повышая производительность поиска.

Рассмотрим пример со следующей структурой документа:

```

1. POST indexname/_doc/
2. {
3.     "product category": "Formal shoes",
4.     "size": 7
5. }
```

Чтобы определить диапазоны размеров, выполним код для агрегации:

```

1. GET indexname/_search
2. {
3.     "aggs": {
```

```

4.     "size_ranges": {
5.       "range": {
6.         "field": "size",
7.         "ranges": [
8.           { "to": 6 },
9.           { "from": 6, "to": 12 },
10.          { "from": 12 }
11.        ]
12.      }
13.    }
14.  }
15. }
```

Результат агрегации вместе с документами будет выглядеть так:

```

1. "aggregations" : {
2.   "size_ranges" : {
3.     "buckets" : [
4.       { "key" : "*-6.0", "to" : 6.0, "doc_count" : 0 },
5.       { "key" : "6.0-12.0", "from" : 6.0, "to" : 12.0, "doc_count" : 5
},
6.       { "key" : "12.0-*", "from" : 12.0, "doc_count" : 2 }
7.     ]
8.   }
9. }
```

Хотя агрегация диапазонов дает всю необходимую информацию о диапазоне размеров, этот процесс можно оптимизировать, включив детализацию диапазонов во время индексирования документов. Таким образом мы избавимся от необходимости последующей агрегации диапазонов. Посмотрим, как добавить эту информацию во время индексирования документа:

```

1. PUT indexname
2. {
3.   "mappings": {
4.     "properties": {
5.       "size_range": {
6.         "type": "keyword"
7.       }
8.     }
9.   }
10. }
```

В примере выше мы создаем маппинг для поля `size_range`, чтобы хранить диапазон размеров, а также название и размер обуви. Затем мы добавим документы в индекс вместе с полем `size_range`:

```

1. POST indexname/_doc/
2. {
3.   "product category": "Formal shoes",
```

```

4.   "size": 5,
5.   "size_range": "0-5"
6. }
```

Мы включаем поле `size_range` для хранения диапазона размеров обуви, что позволяет избежать агрегации диапазонов. Вместо нее можно провести агрегацию терминов, чтобы быстро выводить диапазоны размеров, выбирая уникальные значения из поля `size_range`. Агрегацию диапазона размеров можно выполнить с помощью следующего запроса:

```

1. GET indexname/_search
2. {
3.   "aggs": {
4.     "size_ranges": {
5.       "terms": {
6.         "field": "size_range"
7.       }
8.     }
9.   }
10. }
```

Ответ на запрос будет таким:

```

1. "aggregations" : {
2.   "size_ranges" : {
3.     "doc_count_error_upper_bound" : 0,
4.     "sum_other_doc_count" : 0,
5.     "buckets" : [
6.       { "key" : "0-5", "doc_count" : 1 },
7.       { "key" : "10-15", "doc_count" : 1 },
8.       { "key" : "6-10", "doc_count" : 1 }
9.     ]
10.   }
11. }
```

Предварительное индексирование релевантных данных позволяет избежать дополнительных затрат на операции поиска. Определение полей, которые можно включить наряду с фактическими данными документа, позволяет эффективно извлекать данные и повышать производительность поиска.

Маппинг идентификаторов в качестве ключевых слов

Для некоторых числовых полей, таких как идентификаторы продуктов или блогов, числовые операции обычно не производятся. Поэтому вместо того, чтобы задавать эти поля как числовые, которые Elasticsearch оптимизирует для запросов диапазона, практичнее определять их как ключевые слова (`keyword`). Это позволит эффективно выполнять запросы тех терминов, которые обычно используются для полей идентификаторов. Запросы терминов к полям ключевых

слов выполняются быстрее, чем к числовым полям. Такой стратегический выбор типа данных помогает настроить производительность поиска в Elasticsearch, обеспечивая оптимизацию операций поиска с идентификаторами.

Принудительное слияние индексов, доступных только для чтения

Для дальнейшего повышения производительности поиска можно использовать концепцию объединения шардов, доступных только для чтения, в один сегмент внутри индекса. Объединяя в один сегмент шарды, доступные только для чтения, мы получаем более простую и эффективную структуру данных для операций поиска. Эта оптимизация особенно эффективна для индексов, основанных на времени, в которые нельзя добавлять новые документы после завершения текущего интервала. Поскольку такие индексы обычно доступны только для чтения, к ним можно смело применять операцию принудительного слияния.

ПРИМЕЧАНИЕ Принудительное слияние не следует выполнять для индекса, который открыт для записи или будет открыт в ближайшее время, так как это может привести к нежелательным последствиям.

Использование фильтра вместо запроса

Для получения искомых документов в Elasticsearch можно использовать либо запросы, либо условия фильтрации. Условия запроса позволяют оценить релевантность, чтобы определить, насколько документ соответствует параметрам запроса; а термины фильтра используются для оценки соответствия документа запросу без оценки релевантности. Преимущество использования фильтров заключается в том, что Elasticsearch не оценивает релевантность для условий фильтра, и это сокращает время выполнения. Кроме того, результаты фильтрации можно кэшировать для еще большего увеличения производительности. Поэтому если оценка релевантности не является решающим фактором для конкретного запроса, рекомендуется использовать фильтры вместо запросов, чтобы оптимизировать поиск и обеспечить более быстрое получение результатов.

Увеличение количества реплик

Чтобы повысить производительность поиска в кластере Elasticsearch, можно увеличить количество реплик для данного индекса. Для обработки операций поиска Elasticsearch использует как первичный шард, так и шарды реплик. Увеличивая количество реплик, мы эффективно задействуем больше узлов для поиска. Такой распределенный подход распараллеливает поиск, позволяя

быстрее реагировать на поисковые запросы. Стратегически изменения количество реплик, можно оптимизировать производительность поиска в кластере Elasticsearch, обеспечивая его эффективность и масштабируемость.

Извлечение только необходимых полей

Чтобы оптимизировать производительность поиска, рекомендуется извлекать только необходимые поля, а не все поля индекса. Рассмотрим следующий пример документа в индексе:

```

1. {
2.   "_index": "indexname",
3.   "_type": "_doc",
4.   "_id": "70ss0HIBCxugYrgStg5D",
5.   "_score": 1.0,
6.   "_source": {
7.     "product_category": "Formal shoes",
8.     "size": 14,
9.     "size_range": "10-15"
10.   }
11. }
```

Предположим, что для запроса требуется извлечь лишь поле "product_category". Изменим поисковый запрос, чтобы получить только это конкретное поле:

```

1. GET indexname/_search
2. {
3.   "_source": "product_category",
4.   "query": {
5.     "term": {
6.       "size": 14
7.     }
8.   }
9. }
```

Ответ на этот запрос будет следующим:

```

1. {
2.   "_index": "indexname",
3.   "_type": "_doc",
4.   "_id": "70ss0HIBCxugYrgStg5D",
5.   "_score": 1.0,
6.   "_source": {
7.     "product_category": "Formal shoes"
8.   }
9. }
```

Указав в параметре "_source" нужное имя поля, можно ограничить получаемые данные полем "product_category". Такой подход минимизирует

накладные сетевые расходы, сокращает объем передачи данных и повышает производительность поиска за счет исключения ненужной обработки нерелевантных полей.

Извлекая только необходимые поля, можно оптимизировать использование ресурсов и повысить эффективность поиска в Elasticsearch.

Использование более быстрого оборудования

Для повышения производительности поиска (как и производительности индексирования) рекомендуется использовать более быстрые накопители, например **твердотельные (SSD)**. SSD-накопители обеспечивают более высокую производительность по сравнению с традиционными вращающимися дисками. Также рекомендуется использовать локальные хранилища, а не удаленные файловые системы, такие как SMB или NFS. Хотя облачные хранилища, такие как AWS Elastic Block Storage, удобны и быстры, скорость текущих процессов в них обычно медленнее. Поэтому использование локальных хранилищ может значительно повысить производительность поиска в Elasticsearch.

Выделение памяти для кэша файловой системы

Ускорение операций ввода-вывода с помощью кэша файловой системы обеспечивает преимущество в производительности за счет использования памяти в качестве буфера. Выделив часть доступной памяти на узле Elasticsearch под кэш файловой системы, можно оптимизировать кэширование часто используемых данных. Обычно под кэш файловой системы выделяют примерно половину доступной памяти. Такой подход повышает скорость поиска данных, поскольку часто используемые данные остаются доступны в памяти, что снижает необходимость операций ввода-вывода на диске.

Отказ от стоп-слов при поиске

При составлении поисковых запросов рекомендуется исключать стоп-слова, поскольку они могут существенно повлиять на количество результатов. Стоп-слова — это слова, которые часто встречаются в документах, такие как *the*, *is* или *and*. Включение стоп-слов в поисковый запрос может привести к увеличению набора результатов, поскольку они присутствуют в большинстве документов и не влияют на релевантность результатов поиска. Elasticsearch генерирует оценки для поисковых запросов, и присутствие стоп-слов добавляет нагрузку на обработку, потенциально замедляя производительность. Если необходимо получить результаты по такому запросу, как *the dog*, рекомендуется использовать оператор AND между стоп-словом и реальным поисковым термином

(*the AND dog*), чтобы получить желаемое совпадение без увеличения количества результатов. Отказ от стоп-слов в поисковых запросах позволит повысить производительность поиска в Elasticsearch.

Отказ от скриптовых запросов

Рекомендуется по возможности избегать использования скриптовых запросов в Elasticsearch, так как они негативно влияют на производительность. Рассмотрим пример, в котором нужно найти все документы в индексе с категорией товара, начинающейся с "For". Вот исходный скриптовый запрос:

```

1. GET index/_search
2. {
3.   "query": {
4.     "bool": {
5.       "filter": [
6.         {
7.           "script": {
8.             "script": {
9.               "source": "doc['product category.keyword'].value.
startsWith('For')"
10.            }
11.          }
12.        ]
13.      }
14.    }
15.  }
16. }
```

Скриптовый запрос вернет все совпадающие документы, в которых категория продукта начинается с "For". Вот пример документа, который будет возвращен скриптовым запросом:

```

1. {
2.   "_index" : "index",
3.   "_type" : "_doc",
4.   "_id" : "70ss0HIBCxugYrgStg5D",
5.   "_score" : 0.0,
6.   "_source" : {
7.     "product category" : "Formal shoes",
8.     "size" : 14,
9.     "size_range" : "10-15"
10.   }
11. }
```

Однако использование скриптовых запросов может расходовать много ресурсов и замедлять процесс поиска. Чтобы повысить производительность запросов и избежать использования скриптов, можно использовать альтернативные

типы запросов. Например, префиксные запросы, которые позволяют достичь того же результата без использования скриптов. Вот вариант с использованием префиксного запроса:

```
1. GET index/_search
2. {
3.   "query": {
4.     "prefix" : { "product category.keyword": "For" }
5.   }
6. }
```

Префиксный запрос позволяет получить те же результаты, что и скриптовый, но более эффективным способом. Эта техника оптимизации помогает повысить производительность запросов Elasticsearch, устранив необходимость в ресурсозатратном выполнении скриптов.

ПРИМЕЧАНИЕ Чтобы повысить производительность и эффективность запросов в Elasticsearch, рекомендуется использовать типы запросов, которые лучше подходят для конкретных требований поиска.

НАСТРОЙКА ELASTICSEARCH ДЛЯ ИСПОЛЬЗОВАНИЯ ДИСКА

Оптимизация использования диска — важнейшее условие поддержания производительности и эффективности кластера Elasticsearch. Применяя различные техники для уменьшения объема использования диска, можно обеспечить оптимальное использование ресурсов и избежать потенциально узких мест. Рассмотрим некоторые стратегии настройки Elasticsearch для минимизации потребления дискового пространства.

Сжатие индекса

API сжатия индекса предоставляет возможность уменьшить количество шардов в индексе, что делает структуру индекса более компактной и снижает объем использования дискового пространства. Этот API создает новый целевой индекс с меньшим количеством первичных шардов путем слияния существующих. Однако перед применением операции сжатия необходимо выполнить несколько условий:

- **Индекс только для чтения** — для успешного выполнения операции сжатия исходный индекс должен иметь состояние «только для чтения».
- **Условие одного узла** — для эффективного слияния копии каждого шарда должны находиться на одном узле.

- **Состояние кластера** — перед началом операции сжатия убедитесь, что состояние кластера зеленое.

Чтобы сжать индекс, используется API сжатия с указанием параметров целевого индекса, таких как количество реплик, количество шардов и кодек индекса. Уменьшение количества шардов позволяет значительно оптимизировать использование дискового пространства и повысить общую производительность. Следующий код показывает пример работы API сжатия:

```
1. POST /source_index/_shrink/target_index
2. {
3.   "settings": {
4.     "index.number_of_replicas": 1,
5.     "index.number_of_shards": 1,
6.     "index.codec": "best_compression"
7.   }
8. }
```

В этом примере мы сокращаем исходный индекс `source_index` до целевого индекса `target_index`. Параметры, указанные в теле запроса, определяют конфигурацию целевого индекса, включая количество реплик, количество шардов и кодек индекса.

Принудительное слияние

Elasticsearch хранит данные в шардах, состоящих из нескольких сегментов. Сегменты представляют собой физические файлы на диске, которые содержат фактические данные индекса. В целях эффективности лучше иметь несколько больших сегментов, а не множество маленьких. API принудительного слияния позволяет вручную запускать объединение сегментов, уменьшая их количество в шардах.

Хотя Elasticsearch автоматически выполняет слияние в фоновом режиме, в определенных сценариях рекомендуется вручную инициировать принудительное слияние. Операция принудительного слияния должна выполняться только после завершения записи данных в индекс. Это гарантирует, что все сегменты будут объединены правильно, оптимизируя использование диска.

Чтобы инициировать эту операцию, можно использовать API принудительного слияния как для определенного индекса, так и для всех индексов в кластере. Важно отметить, что принудительное слияние следует выполнять с осторожностью и не слишком часто, поскольку оно может требовать большого количества ресурсов.

Чтобы принудительно объединить индекс, нужно выполнить следующую команду:

```
POST /indexname/_forcemerge
```

Эта команда запускает операцию принудительного слияния для указанного имени индекса. Применяя конечную точку `_forcemerge` к индексу, мы можем объединить осколки и уменьшить количество сегментов, что приведет к оптимизации использования диска.

Чтобы принудительно объединить все индексы, нужно выполнить следующую команду:

```
POST /_forcemerge
```

Эта команда запускает операцию принудительного слияния всех индексов в кластере. Кроме того, можно принудительно объединить конкретные индексы, разделив их запятыми:

```
POST /indexname1,indexname2,indexname3/_forcemerge
```

Отключение необязательных функций

В Elasticsearch можно включать или отключать определенные функции для конкретных полей, исходя из требований к индексированию и запросам. Отключение ненужных функций может значительно оптимизировать использование диска и повысить производительность запросов. Рассмотрим некоторые варианты отключения ненужных функций в Elasticsearch.

Отключение индексации полей

Не все поля в документах нужно индексировать. Иногда достаточно только хранить и извлекать их значения, не выполняя при этом операций поиска или фильтрации. Для таких полей индексирование можно отключить с помощью параметра `"index"`. Вот пример:

```
1. PUT indexname
2. {
3.   "mappings": {
4.     "properties": {
5.       "product_code": {
6.         "type": "integer",
7.         "index": false
8.       }
9.     }
10.   }
11. }
```

В примере выше мы определяем маппинг для поля `"product_code"` и устанавливаем для свойства `"index"` значение `false`. Оно сообщает Elasticsearch, что индексировать это поле не требуется, что экономит дисковое пространство и снижает накладные расходы на индексирование.

Отключение норм для текстовых полей

Нормы используются для расчета оценки релевантности документа при полно-текстовом поиске. Однако в некоторых сценариях оценка по определенным текстовым полям не требуется. Для таких полей нормы можно отключить, чтобы оптимизировать использование диска. Вот пример:

```

1. PUT indexname
2. {
3.   "mappings": {
4.     "properties": {
5.       "color": {
6.         "type": "text",
7.         "norms": false
8.       }
9.     }
10.   }
11. }
```

В запросе выше мы установили значение `false` для свойства `"norms"` поля `"color"`, это означает, что оценка по этому полю не требуется. Отключение норм позволяет сэкономить дисковое пространство и повысить производительность индексирования.

Отключение позиций для текстовых полей

По умолчанию Elasticsearch хранит информацию о позициях для текстовых полей, которая используется для фразовых запросов и оценки релевантности на основе близости. Однако если по определенному полю фразовые запросы не требуются, информацию о позициях можно отключить, чтобы оптимизировать использование диска. Вот пример:

```

1. PUT indexname
2. {
3.   "mappings": {
4.     "properties": {
5.       "color": {
6.         "type": "text",
7.         "index_options": "freqs"
8.       }
9.     }
10.   }
11. }
```

В примере выше мы установили для свойства `"index_options"` поля `"color"` значение `"freqs"`. Это отключает информацию о позиции, но при этом сохраняет информацию о частоте терминов. Это может быть полезно, если для поля не требуются фразовые запросы.

Выборочно отключая индексацию, нормы и информацию о позиции для определенных полей, можно точно настроить маппинг и оптимизировать использование диска в Elasticsearch. Важно тщательно оценить требования создаваемого приложения и выбрать подходящие настройки для каждого поля, чтобы найти баланс между функциональностью и производительностью.

Отказ от динамического маппинга строк

В Elasticsearch строковые поля по умолчанию динамически отображаются с типом `text` (текстовый) и `keywords` (ключевые слова). Однако такой двойной маппинг не всегда необходим и может привести к лишнему использованию диска. Важно учитывать специфические требования к каждому строковому полю и избегать накладных расходов на двойной маппинг, если этого не требуется.

Например, такие поля, как поле ID (идентификатор), обычно требуют со-впадения и не нуждаются в поддержке полнотекстового поиска. С другой стороны, для таких полей, как «тело», желательно обеспечить возможность полнотекстового поиска.

Чтобы избежать двойного маппинга и оптимизировать использование диска, можно явно определить маппинг для каждого поля или создать шаблон индекса, который будет применять нужные настройки маппинга.

Рассмотрим пример, в котором поле ID продукта сопоставляется с ключевым словом:

```
1. PUT my_index
2. {
3.   "mappings": {
4.     "properties": {
5.       "product_code": {
6.         "type": "keyword"
7.       }
8.     }
9.   }
10. }
```

Здесь для поля кода продукта `product_code` явно задается тип ключевого слова. Это гарантирует, что поле будет проиндексировано и сохранено как ключевое слово, что позволит эффективно искать точное соответствие.

Используя явный маппинг или шаблоны индексов, можно избежать ненужного двойного маппинга строковых полей и оптимизировать использование диска в Elasticsearch. Важно тщательно продумать требования к каждому полю и выбрать подходящий тип маппинга, чтобы добиться желаемого поведения при поиске и индексировании.

Отключение `_source`

В Elasticsearch поле `_source` в ответе содержит фактическое JSON-тело документа. Однако документ JSON в ответе на поисковый запрос может не понадобиться, и отказ от него позволит избежать лишней загрузки полосы пропускания и повысить производительность. В таких случаях мы можем отключить поле `_source`.

Чтобы сделать это, изменим маппинг индекса, как в следующем примере:

```
1. PUT indexname
2. {
3.   "mappings": {
4.     "_source": {
5.       "enabled": false
6.     }
7.   }
8. }
```

Мы отключили поле `_source`, установив для свойства `"enabled"` значение `false` в маппинге индекса.

После отключения поля `_source` при добавлении документов в индекс ответ на поисковый запрос больше не будет включать JSON-документ. Он будет представлять информацию о просмотрах, не раскрывая содержимого документа.

Теперь добавим в индекс несколько документов, чтобы понять, как это изменение повлияет на выдачу при поиске документов. Вот пример:

```
1. POST indexname/_doc/
2. }
3.   "product category": "Formal shoes",
4.   "size": 5,
5.   "size_range": "0-5"
6. }
```

В этом примере показано создание документа, и таким же способом можно добавлять другие документы в индекс. Теперь осуществим поиск по индексу, возвращающий документ. Для этого выполним следующую команду:

```
GET indexname/_search
```

Ответ будет выглядеть так:

```
1. "hits" : [
2.   {
3.     "_index" : "indexname",
4.     "_type" : "_doc",
5.     "_id" : "ZKbrRnIBzbesIqv89MUZ",
```

```
6.      "_score" : 1.0
7.    }
8. ]
```

Видим, что в ответе отсутствует поле `_source`, но мы по-прежнему имеем доступ к метаданным, таким как индекс, тип, ID и оценка соответствия для каждого совпадения.

Отключив поле `_source`, можно оптимизировать пропускную способность сети и производительность поиска в сценариях, когда нужно работать только с метаданными или определенными полями документов, а все тело JSON не требуется.

Оптимизация типов числовых полей

Чтобы оптимизировать использование диска в Elasticsearch, важно выбрать подходящий числовой тип данных для их эффективного хранения. Используя наименьший достаточный тип для размещения данных, можно минимизировать использование дискового пространства и повысить общую производительность.

Для целочисленных значений подойдут типы `byte`, `short`, `integer` и `long`. Если данные попадают в диапазон меньшего типа, достаточно использовать его, а не больший. Точно так же для значений с плавающей точкой можно выбрать тип `float`, `double` или `half_float` в зависимости от точности, требуемой для данных.

Выбрав подходящий числовой тип, можно эффективно управлять дисковым пространством и не тратить память на типы, вмещающие больший диапазон значений, чем это нужно.

ЛУЧШИЕ ПРАКТИКИ ELASTICSEARCH

В дополнение к конкретным способам оптимизации, рассмотренным выше, существуют общие лучшие практики, которых следует придерживаться, чтобы обеспечить оптимальную производительность и избежать потенциальных проблем при использовании Elasticsearch. Рассмотрим некоторые из этих практик.

Явное определение маппинга индексов Elasticsearch

Определение маппинга индекса Elasticsearch — важнейшая из лучших практик. Прежде чем вводить документы в Elasticsearch, важно четко представить данные и их структуру. Определив маппинг заранее, можно явно указать типы данных и свойства для каждого поля в индексе.

Автоматический маппинг Elasticsearch определяет типы данных на основе первого проиндексированного документа. Это может быть удобно, но также может привести к неожиданным проблемам. Elasticsearch может сделать неверные предположения о типе данных, что в дальнейшем приведет к несоответствиям и проблемам с запросами.

Определяя маппинг самостоятельно, мы получаем полный контроль над интерпретацией и индексированием данных. Мы можем точно указать, какое поле следует рассматривать как строку, число, дату или любой другой тип данных. Кроме того, мы можем определить такие свойства, как параметры индекса, анализаторы и пользовательские настройки, которые соответствуют конкретным требованиям к данным.

Явный маппинг не только помогает обеспечить целостность и согласованность данных, но и открывает возможности для настройки производительности. Некоторые методы оптимизации, такие как отключение индексации для определенных полей или использование соответствующих типов данных, возможны только при явном маппинге.

Затраты времени на определение маппинга до получения данных окупаются предотвращением потенциальных конфликтов типов данных и гарантией того, что Elasticsearch будет вести себя предсказуемо. Это основа для эффективного запроса, индексирования и поиска в кластере Elasticsearch.

Оптимизация производительности кластера Elasticsearch

Планирование мощности — важнейший шаг оптимизации производительности Elasticsearch и обеспечения стабильной и эффективной работы кластера. Оно включает в себя оценку различных параметров, таких как дисковое пространство, использование памяти и ресурсов процессора для удовлетворения потребностей в хранении, индексировании и поиске данных. Вот что необходимо учитывать для эффективного планирования:

- **Период хранения данных.** Определите, как долго нужно хранить данные в кластере Elasticsearch. Это поможет оценить необходимое дисковое пространство и учесть рост объема данных с течением времени.
- **Скорость индексирования данных.** Оцените скорость, с которой новые данные будут индексироваться в Elasticsearch. Учитывайте такие факторы, как объем данных, частота обновлений и ожидаемая пропускная способность индексирования. Это поможет определить необходимую мощность для индексирования при обработке поступающих данных.

- **Скорость поиска данных.** Проанализируйте ожидаемую поисковую нагрузку на кластер Elasticsearch. Определите частоту и сложность поисковых запросов, которые будут выполняться. Это повлияет на то, какие ресурсы потребуются для эффективной работы поиска.
- **Количество реплик.** Определите желаемый уровень избыточности данных и высокой доступности. Реплики — это дополнительные копии каждого шарда, которые могут обслуживать запросы на чтение и обеспечивать отказоустойчивость. Определите количество реплик, необходимое для удовлетворения требований к доступности, и учтите дополнительные затраты ресурсов на них.

Планирование мощности — это итеративный процесс, который предполагает постоянную корректировку на основе бенчмаркинга и мониторинга производительности кластера. При изменении моделей использования или увеличении объемов данных может потребоваться скорректировать настройки. Elasticsearch предлагает горизонтальную масштабируемость, позволяющую добавлять в кластер дополнительные узлы для распределения рабочей нагрузки и удовлетворения растущих потребностей в мощности.

Тщательное планирование мощности позволит убедиться, что кластер Elasticsearch хорошо обеспечен ресурсами и способен эффективно обслуживать запросы пользователей. Это поможет избежать нехватки ресурсов, узких мест в производительности и потенциальных простоев, обеспечивая плавное и надежное развертывание Elasticsearch.

Как избежать проблемы **split-brain**

В распределенном кластере Elasticsearch несколько узлов работают совместно, образуя единый кластер и обеспечивая масштабируемость, отказоустойчивость и высокую доступность. Однако существует потенциальная проблема, называемая **ситуацией разделения мозга (split-brain)**, которая может возникнуть в распределенных системах, включая Elasticsearch.

Разделение мозга возникает, когда в кластере образуется разветвление сети или обрыв связи, в результате чего узлы теряют связь друг с другом.

В таком случае узлы могут ошибочно предположить, что кластер разделился на отдельные части, каждая из которых имеет свой главный узел. Это может привести к несогласованности данных, конфликтам и проблемам в работе.

Чтобы предотвратить эту проблему, Elasticsearch предоставляет механизм, известный как **выбор главного узла**. Когда кластер обнаруживает разделение, узлы, которые еще поддерживают взаимную коммуникацию, начинают

выбирать новый главный узел. Это гарантирует сохранение только одного активного главного узла, отвечающего за координацию кластера.

Для настройки Elasticsearch так, чтобы избежать проблемы разделения мозга, можно использовать параметр `discovery.zen.minimum_master_nodes`. Установив значение этого параметра равным или большим, чем половина количества узлов в кластере плюс один, вы обеспечите достаточное большинство узлов для избрания нового главного. Например, если в кластере три узла, установка значения параметра `minimum_master_nodes` равным двум гарантирует, что в процессе выбора будут участвовать минимум два из трех узлов, что позволит избежать ситуации разделения мозга.

Правильная настройка параметра `minimum_master_nodes` снижает риск возникновения сценариев с разделением мозга и сохраняет целостность и стабильность кластера Elasticsearch. Очень важно учитывать этот параметр при настройке кластера, чтобы обеспечить его последовательную и надежную работу.

Включение лога медленных запросов

Для мониторинга и оптимизации производительности запросов в Elasticsearch важно включить лог медленных запросов. Этот лог помогает выявлять запросы, которые выполняются дольше заданного порога, чтобы определить узкие места в производительности и принять необходимые меры для оптимизации выполнения запросов.

Включение лога медленных запросов требует настройки порога длительности для разных уровней лога. Задавая пороговые значения, Elasticsearch определяет, когда время выполнения запроса превышает заданное и должно быть занесено в лог. Пороговые значения для уровней предупреждения, информации, отладки и трассировки могут различаться в зависимости от детализации, необходимой для мониторинга.

Вот пример, демонстрирующий, как включить лог медленных запросов и установить пороговые значения длительности:

```
1. PUT /index_name/_settings
2. {
3.   "index.search.slowlog.threshold.query.warn": "10s",
4.   "index.search.slowlog.threshold.query.info": "5s",
5.   "index.search.slowlog.threshold.query.debug": "2s",
6.   "index.search.slowlog.threshold.query.trace": "500ms",
7.   "index.search.slowlog.threshold.fetch.warn": "1s",
8.   "index.search.slowlog.threshold.fetch.info": "800ms",
9.   "index.search.slowlog.threshold.fetch.debug": "500ms",
10.  "index.search.slowlog.threshold.fetch.trace": "200ms",
11.  "index.search.slowlog.level": "info"
12. }
```

В этом запросе мы установили разные пороги длительности для запросов и операций выборки на разных уровнях лога. Например, если длительность запроса превышает 10 секунд, будет вызван лог предупреждения, а запросы длительностью более 5 секунд будут регистрироваться на уровне информации. Так же устанавливаются пороговые значения для уровней логов отладки и трассировки. Параметр `index.search.slowlog.level` задает минимальный уровень лога, на котором будут регистрироваться медленные запросы.

После успешного выполнения запроса конфигурации Elasticsearch начнет регистрировать медленные запросы на основе указанных пороговых значений. Затем записи лога медленных запросов можно анализировать, чтобы выявлять запросы, требующие оптимизации или дальнейшего изучения.

Ответ на запрос конфигурации будет выглядеть так:

```
1. {  
2.   "acknowledged": true  
3. }
```

Этот ответ указывает, что настройка лога медленных запросов прошла успешно. С этого момента Elasticsearch будет регистрировать медленные запросы на основе заданных пороговых значений длительности. Просматривая логи медленных запросов, можно получить полезные сведения о производительности запросов, что позволяет выявить узкие места, оптимизировать выполнение запросов и повысить общую производительность системы.

Включение лога медленных запросов — важная практика мониторинга производительности запросов и выявления областей, требующих оптимизации. Регулярно просматривая и анализируя записи лога, можно точно настроить кластер Elasticsearch, снизить время отклика запросов и обеспечить для пользователей более эффективный поиск.

ЗАКЛЮЧЕНИЕ

В этой главе мы рассмотрели вопросы повышения производительности кластера Elasticsearch. Мы изучили методы тонкой настройки скорости индексирования и поиска, оптимизирующие разные параметры функциональности кластера. Чтобы повысить скорость индексирования, мы сделали акцент на эффективном получении данных с помощью массовых запросов, знаний о распределении шардов, настройки интервалов обновления и использовании более быстрого оборудования, например SSD-накопителей.

Для повышения скорости поиска использовались такие стратегии, как тщательное моделирование документов, ограничение поиска определенными полями,

предварительное индексирование общих запросов, использование фильтров в определенных случаях и увеличение количества реплик для распределения поисковой нагрузки. Методы оптимизации использования диска включали сокращение индексов, принудительное объединение сегментов и минимизацию потребностей в хранении за счет отказа от ненужного динамического маппинга строк.

Мы выделили такие ключевые методы повышения производительности, как явный маппинг полей индексов, тщательное планирование мощности, включение лога медленных запросов и избегание ситуации разделения мозга.

В следующей главе мы перейдем к администрированию кластера Elasticsearch. Мы разберем такие темы, как обеспечение безопасности, создание псевдонимов индексов, управление снапшотами и изучение **общей схемы Elastic (Elastic Common Schema, ECS)** для организованного анализа данных.

ВОПРОСЫ

1. Назовите некоторые эффективные стратегии настройки скорости индексирования в Elasticsearch.
2. Объясните концепцию интервала обновления в Elasticsearch и его значение для оптимизации производительности.
3. Какие существуют методы и подходы для настройки скорости поиска в Elasticsearch?
4. Объясните, в чем состоит различие между фильтрами и запросами в Elasticsearch и когда следует использовать каждый из них.
5. Как в Elasticsearch извлечь только нужные поля, а не документ целиком?
6. Какие методы рекомендуются для оптимизации использования диска в Elasticsearch и минимизации потребления ресурсов?

ГЛАВА 11

Администрирование: управление кластерами Elasticsearch

ВВЕДЕНИЕ

В предыдущей главе мы рассмотрели методы повышения производительности кластера Elasticsearch. Мы начали с оптимизации скорости индексирования и подробно изучили различные стратегии тонкой настройки возможностей поиска в кластере. Кроме того, мы обсудили эффективные методы оптимизации использования диска и поделились лучшими практиками для повышения общей производительности кластера.

В этой главе мы сосредоточимся на задачах администрирования Elasticsearch. Вначале мы рассмотрим меры безопасности и поймем, как эффективно их применять. Затем изучим концепцию псевдонимов индексов, их назначение и создание. Кроме того, мы узнаем о значении снапшотов и репозиториев в Elasticsearch, научимся делать снапшоты индексов и восстанавливать их при необходимости. Наконец, мы рассмотрим **общую схему Elastic (ECS)** и поймем ее важность для стандартизации представления данных в Elasticsearch.

К концу этой главы вы получите полное представление об основных задачах администрирования в Elasticsearch, что позволит вам эффективно управлять кластером Elasticsearch и обеспечивать его безопасность.

СТРУКТУРА

В этой главе:

- Безопасность Elasticsearch
- Псевдонимы индексов
- Создание репозитория и снапшота
- Восстановление снапшота
- Общая схема Elastic (Elastic Common Schema)
- Масштабирование кластера Elasticsearch
- Мониторинг Elasticsearch

ЦЕЛИ

По завершении этой главы вы научитесь настраивать систему безопасности Elasticsearch для защиты кластера. Кроме того, вы научитесь настраивать псевдонимы индексов для повышения гибкости поиска и выполнять необходимые операции со снапшотами для обеспечения резервного копирования и восстановления данных.

БЕЗОПАСНОСТЬ ELASTICSEARCH

Начиная с Elastic Stack версий 6.8 и 7.1, Elasticsearch предоставляет несколько функций безопасности, доступных бесплатно. Эти функции включают шифрование TLS для безопасной коммуникации и **контроль доступа на основе ролей (RBAC)** для тонкой настройки пользовательских разрешений. В этом разделе мы рассмотрим, как настроить TLS-шифрование и аутентификацию в Elasticsearch. Кроме того, мы узнаем, как установить безопасное соединение между Kibana и Elasticsearch.

Предполагая, что у вас уже установлены Elasticsearch и Kibana, основное внимание мы уделим настройке управления доступом на основе ролей с помощью интерфейса Kibana. Хотя устанавливать Kibana не обязательно, ее наличие позволит легко создавать роли и пользователей и назначать им соответствующие разрешения.

Давайте перейдем к настройке TLS на кластере Elasticsearch. В этом примере мы рассмотрим кластер с одним узлом, но ту же конфигурацию можно применить и к нескольким узлам, если кластер многоузловой.

Настройка TLS

Чтобы настроить TLS в Elasticsearch, необходимо создать хранилища ключей и секретов. Хранилище ключей содержит закрытый ключ и сертификат для сервера Elasticsearch, а хранилище секретов — сертификаты для доверенных клиентов.

Для генерирования самозаверенного сертификата, используемого для TLS, выполним команду `elasticsearch-certutil`, как в следующем примере:

```
bin/elasticsearch-certutil cert -out config/elastic-certificates.p12 -pass ""
```

Она создаст файл `elastic-certificates.p12` в каталоге `config`. Этот файл содержит хранилище ключей и хранилище секретов, которые нужны для настройки TLS в Elasticsearch.

После генерирования сертификата необходимо настроить Elasticsearch на его использование. Для этого нужно отредактировать файл `elasticsearch.yml` и установить следующие свойства:

1. `xpack.security.enabled: true`
2. `xpack.security.transport.ssl.enabled: true`
3. `xpack.security.transport.ssl.verification_mode: certificate`
4. `xpack.security.transport.ssl.keystore.path: config/elastic-certificates.p12`
5. `xpack.security.transport.ssl.truststore.path: config/elastic-certificates.p12`

После настройки TLS нужно перезапустить Elasticsearch, чтобы изменения вступили в силу.

Чтобы запустить Elasticsearch, выполните следующую команду:

```
sudo service elasticsearch start
```

После запуска Elasticsearch к нему можно подключиться по протоколу https. Например, чтобы подключиться к Elasticsearch на localhost, используем следующий URL:

```
https://localhost:9200
```

В ответ будет предложено ввести имя пользователя и пароль для пользователя с ролью `elastic`.

Пароли кластера Elasticsearch

После запуска службы Elasticsearch необходимо настроить пароль для кластера Elasticsearch, выполнив описанные ниже действия.

1. Перейдите в каталог Elasticsearch Home и выполните команду

```
bin/elasticsearch-setup-passwords auto
```

2. После выполнения этой команды Elasticsearch запросит подтверждение перед установкой паролей для разных встроенных пользователей, таких как `apm_system`, `kibana`, `logstash_system`, `beats_system`, `remote_monitoring_user` и `elastic user`. Получим следующий ответ:

```
1. Initiating the setup of passwords for reserved users elastic1,  
    apm_system, kibana, logstash_system, beats_system,  
    remote_monitoring_user.  
2. The passwords will be randomly generated and printed to the console2.  
3. Please confirm that you would like to continue3 [y/N]  
4.  
5. Changed password for user4 apm_system  
6. PASSWORD apm_system = ITWroWSyMR1DHDYpzQdP  
7.  
8. Changed password for user kibana  
9. PASSWORD kibana = aiX60BRd1JdgXU4faCyK  
10.  
11. Changed password for user logstash_system  
12. PASSWORD logstash_system = wNAbnUNSy8bMs09vhH62  
13.  
14. Changed password for user beats_system  
15. PASSWORD beats_system = Tmo7y8MjxN93XAPrkC0S  
16.  
17. Changed password for user remote_monitoring_user  
18. PASSWORD remote_monitoring_user = gNkO8z6UhZtkVmzkn922  
19.  
20. Changed password for user elastic  
21. PASSWORD elastic = fP12L4sieawIcaWMxSUE
```

3. Таким образом можно создать автоматические пароли для встроенных пользователей. Чтобы установить пароли вручную, необходимо удалить часть `auto` из команды.
4. После того как пароли будут сгенерированы, мы получим доступ к Elasticsearch и Kibana.

¹ Initiating the setup of passwords for reserved users elastic – Запуск установки паролей для зарезервированных пользователей. – *Примеч. ред.*

² The passwords will be randomly generated and printed to the console – Пароли будут сгенерированы случайным образом и выведены на консоль. – *Примеч. ред.*

³ Please confirm that you would like to continue – Подтвердите, что хотите продолжить. – *Примеч. ред.*

⁴ Changed password for user – Изменение пароля для пользователя. – *Примеч. ред.*

5. Безопасность в Elasticsearch включена, и это можно проверить, выполнив команду

```
curl localhost:9200
```

6. Получим следующий ответ:

```
1. {
2.   "error" : {
3.     "root_cause" : [
4.       {
5.         "type" : "security_exception",
6.         "reason" : "missing authentication credentials for REST
request1 [/?pretty]",
7.         "header" : {
8.           "WWW-Authenticate" : "Basic realm=\"security\""
charset=\"UTF-8\""
9.         }
10.      }
11.    ],
12.    "type" : "security_exception",
13.    "reason" : " missing authentication credentials for REST
request [/?pretty]",
14.    "header" : {
15.      "WWW-Authenticate" : "Basic realm=\"security\" charset=\"UTF-8\""
16.    }
17.  },
18.  "status" : 401
19. }
```

7. Теперь, чтобы получить доступ к Elasticsearch, нужно передать имя пользователя и пароль. Для этого выполните следующую команду:

```
curl elastic:fP12L4sieawIcaWMxSUE@localhost:9200
```

8. Эта команда передает имя пользователя elastic и пароль fP12L4sieawIcaWMxSUE в запросе curl. Получим следующий ответ:

```
1. {
2.   "name" : "MacBook-Pro-10.local",
3.   "cluster_name" : "elasticsearch",
4.   "cluster_uuid" : "1qXgBTiAS528an81yRX2jg",
5.   "version" : {
6.     "number" : "8.7.1",
7.     "build_flavor" : "default",
8.     "build_type" : "tar",
```

¹ missing authentication credentials for REST request — Отсутствуют учетные данные аутентификации для запроса REST. — Примеч. ред.

```

9.   "build_hash" : "f229ed3f893a515d590d0f39b05f68913e2d9b53",
10.  "build_date" : "2023-04-27T04:3 3:42.127815583Z",
11.  "build_snapshot" : false,
12.  "lucene_version" : "9.5.0",
13.  "minimum_wire_compatibility_version" : "7.17.0",
14.  "minimum_index_compatibility_version" : "7.0.0"
15. },
16.  "tagline" : "You Know, for Search"
17. }

```

Таким образом настраивается безопасность в Elasticsearch.

После включения безопасности в Elasticsearch также необходимо настроить Kibana для работы с защищенным кластером Elasticsearch. Если попытаться запустить Kibana без каких-либо изменений, получим следующее сообщение:

```
Kibana server is not ready yet1
```

Чтобы настроить Kibana с именем пользователя и паролем Elasticsearch, необходимо обновить файл `kibana.yml`, расположенный по адресу `/etc/kibana/` на Ubuntu. Откройте файл `kibana.yml` и раскомментируйте строки имени пользователя и пароля. Затем установите в качестве пароля вновь созданный пароль. Вот пример фрагмента из файла `kibana.yml`:

```

1. elasticsearch.username: "kibana"
2. elasticsearch.password: "aiX60BRdlJdgXU4faCyK"

```

После изменения имени пользователя и пароля Elasticsearch необходимо перезапустить Kibana с помощью следующей команды:

```
sudo service kibana restart
```

Зайдя в Kibana через URL-адрес `http://localhost:5601`, увидим экран входа в систему (рис. 11.1).

Теперь для входа в Kibana можно использовать имя пользователя и пароль, созданные с помощью инструмента Elasticsearch.

¹ Сервер Kibana не готов. — Примеч. ред.

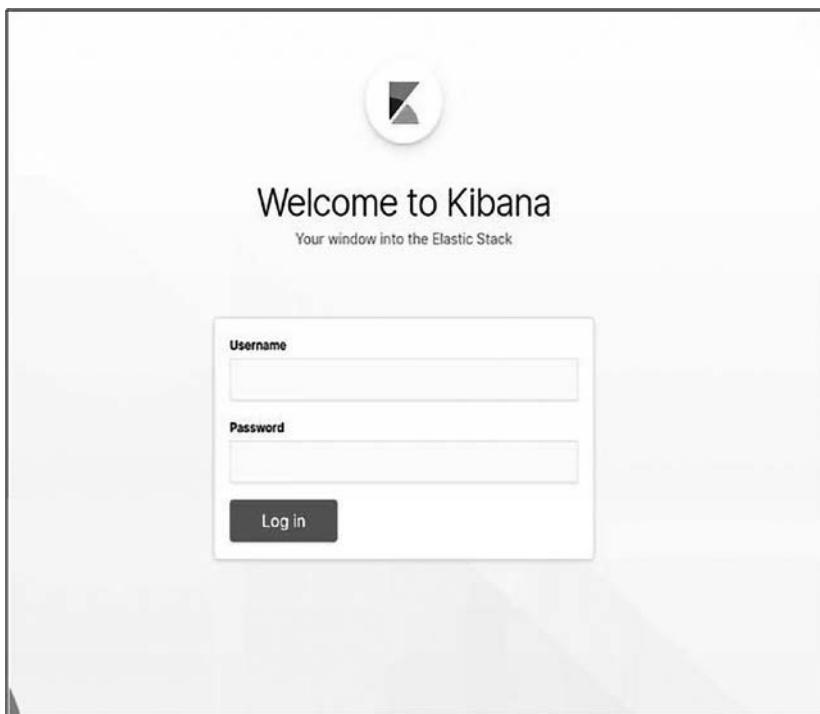


Рис. 11.1. Экран входа в систему Kibana

Настройка доступа на основе ролей с помощью Kibana

Разберемся, как настраивать различные роли и назначать роли пользователям с помощью интерфейса Kibana. Начнем с создания пользователей.

Создание пользователей

Рассмотрим, как создавать пользователей, поскольку это необходимо, если требуется предоставить доступ к Kibana разным людям. Доступ к различным индексам и другим компонентам можно ограничить для разных пользователей на основе их ролей, о чем мы расскажем ниже. Чтобы создать пользователей в Kibana, необходимо сделать следующее:

1. После входа в Kibana нажмите ссылку **Management** (Управление) в левом меню, чтобы открыть экран управления Kibana.

2. Затем нажмите ссылку **Users** (Пользователи) в разделе **Security** (Безопасность).

Откроется окно **Users** со списком существующих, предварительно заданных пользователей Elasticsearch, таких как `elastic`, `kibana`, `logstash_system`, `beats_system` и т. д. (рис. 11.2).

The screenshot shows a table titled 'Users' with the following data:

User Name	Full Name	Email Address	Roles	Status
<code>apm_system</code>			<code>apm_system</code>	Reserved
<code>beats_system</code>			<code>beats_system</code>	Reserved
<code>elastic</code>			<code>superuser</code>	Reserved
<code>kibana</code>			<code>kibana_system</code>	Reserved
<code>kibana_system</code>			<code>kibana_system</code>	Deprecated
<code>logstash_system</code>			<code>logstash_system</code>	Reserved
<code>remote_monitoring_user</code>			<code>remote_monitoring_collector</code> <code>remote_monitoring_agent</code>	Reserved

Рис. 11.2. Список пользователей Kibana

- Чтобы создать нового пользователя, нажмите кнопку **Create user** (Создать пользователя) в правом верхнем углу.
- Откроется форма регистрации нового пользователя, где можно добавить имя пользователя (**Username**), пароль (**Password**), полное имя (**Full name**), адрес электронной почты (**Email address**) и т. д., а также назначить одну или несколько ролей (рис. 11.3).
- После заполнения формы нажмите кнопку **Create user**, чтобы создать пользователя. После этого снова откроется страница со списком пользователей, где мы увидим только что созданного пользователя.
- Пользователя можно отредактировать, нажав на ссылку имени пользователя на экране списка пользователей.
- Пользователя можно удалить, отметив его флагжком, а затем нажав на кнопку **Delete user** (Удалить пользователя). Также можно удалить пользователя с экрана редактирования, нажав на кнопку **Delete user**.

Мы узнали, как создавать пользователей на странице управления Kibana. Теперь разберемся, как создавать роли.

Create user

Profile
Provide personal details.

Username

Full name

Email address

Password
Protect your data with a strong password.

Confirm password

Privileges
Assign roles to manage access and permissions.

Roles

Select roles

Learn what privileges individual roles grant.

Create user **Cancel**

Рис. 11.3. Экран создания пользователя Kibana

Создание ролей

Можно создавать разных пользователей и назначать им разные роли. У ролей могут быть разные разрешения, которые служат для ограничения доступа пользователей, например, к выбранным индексам Elasticsearch. Чтобы создать роли, необходимо сделать следующее:

1. После входа в Kibana нажмите на ссылку **Management** в левом меню, чтобы открыть экран управления.
2. Затем нажмите на ссылку **Roles** (Роли) в разделе **Security** (Безопасность). Откроется страница ролей, содержащая список всех существующих предварительно заданных ролей (рис. 11.4).
3. Чтобы создать новую роль, необходимо нажать кнопку **Create role** (Создать роль) в правом верхнем углу.
4. Откроется страница создания роли (**Create role**), где можно настроить имя роли (**Role name**), параметры роли, такие как привилегии кластера (**Cluster**

privileges), привилегии запуска (Run As privileges) и привилегии индекса (Index privileges). Также можно настроить привилегии доступа к Kibana (рис. 11.5).

Рис. 11.4. Экран списка ролей в Kibana

Рис. 11.5. Экран создания роли в Kibana

5. После настройки разрешений роли можно нажать кнопку *Create role*, чтобы сохранить роль. На странице списка пользователей можно редактировать пользователя и назначать ему роли.

ПСЕВДОНИМЫ ИНДЕКСА

В Elasticsearch можно назначить индексу вторичное имя, или псевдоним. Сопоставляя индекс с псевдонимом, можно выполнять различные запросы к индексу, используя псевдоним. Также можно сопоставить несколько индексов с одним и тем же псевдонимом. Это может быть полезно для разных целей, например:

- **Переиндексация без простоев.** Как уже говорилось, псевдонимы индексов позволяют выполнять переиндексацию без простоев. Это делается благодаря возможности сопоставить псевдоним с индексом, а затем переиндексировать индекс без изменения псевдонима. Таким образом, приложение может продолжать использовать псевдоним даже во время переиндексации.
- **Мониторинг нескольких индексов.** Псевдонимы индексов можно использовать для мониторинга нескольких индексов, как если бы они были одним индексом. Это может быть полезно, если требуется отследить общее состояние группы индексов или создать единый дашборд, на котором будут отображаться данные из нескольких индексов.
- **Маршрутизация поисковых запросов.** Псевдонимы индексов можно использовать, чтобы направлять поисковые запросы в разные индексы. Это может быть полезно, если требуется распределить поисковый трафик по нескольким индексам или обеспечить выполнение определенных типов поисковых запросов только в определенных индексах.

Чтобы создать или удалить псевдоним индекса, используется конечная точка `_aliases`. В следующем примере показано, как создать псевдоним `users_alias`, указывающий на индекс `userdetails`:

```
1. POST /_aliases
2. {
3.   "actions": [
4.     {
5.       "add": {
6.         "index": "userdetails",
7.         "alias": "users_alias"
8.       }
9.     }
10.   ]
11. }
```

Мы создали псевдоним `users_alias`, используя индекс `userdetails`. На псевдоним индекса можно ссылаться так же, как и на имя самого индекса. Например, следующий поисковый запрос будет искать в индексе `users_alias`:

```
GET /users_alias/_search
```

Созданный псевдоним можно удалить с помощью следующего выражения:

```
1. POST /_aliases
2. {
3.   "actions": [
4.     {
5.       "remove": {
6.         "index": "userdetails",
7.         "alias": "users_alias"
8.       }
9.     }
10.   ]
11. }
```

Мы удалили псевдоним `"users_alias"` из индекса `"userdetails"`. Таким образом, можно добавлять или удалять псевдонимы из одного или нескольких индексов. Чтобы связать несколько индексов с одним именем псевдонима, используется следующее выражение:

```
1. POST /_aliases
2. {
3.   "actions": [
4.     {
5.       "add": {
6.         "indices": ["userdetails", "user_details"],
7.         "alias": "users_alias"
8.       }
9.     }
10.   ]
11. }
```

После выполнения этого кода псевдоним будет представлять собой объединенные результаты более чем одного индекса.

Теперь обсудим, как псевдонимы индексов могут обеспечить переиндексацию с нулевым временем простоя. Предположим, у нас есть индекс под названием `"index1"`, обслуживающий приложение, и мы хотим переиндексировать его в `"index2"`, избежав при этом сбоев в работе приложения. Воспользуемся псевдонимами и выполним следующие действия:

1. Создадим псевдоним и сопоставим его с `"index1"`. Этот псевдоним служит для устранения зависимости от имени `"index1"`.

2. Сопоставим приложение с псевдонимом вместо имени "index1".
3. Выполним переиндексацию с "index1" на "index2".
4. После завершения переиндексации удалим сопоставление псевдонимов с "index1" и сопоставим его с "index2". Это можно сделать с помощью кода, приведенного ниже:

```
1. POST /_aliases
2. {
3.   "actions": [
4.     {
5.       "remove": {
6.         "indices": ["index1"],
7.         "alias": "users_alias"
8.       }
9.     },
10.    {
11.      "add": {
12.        "indices": ["index2"],
13.        "alias": "users_alias"
14.      }
15.    }
16.  ]
17. }
```

В этом примере мы удалили сопоставление псевдонимов из "index1" и связали его с "index2". Такой подход позволяет провести переиндексацию, не внося изменений в работающее приложение, что гарантирует отсутствие простоев.

СОЗДАНИЕ РЕПОЗИТОРИЯ И СНАПШОТА

В Elasticsearch снапшот — это резервная копия всего кластера или отдельных индексов, которую можно использовать для восстановления данных в случае сбоя. Снапшоты могут храниться на локальных машинах или удаленно, например в S3, облачном хранилище Google или облаке Azure. Резервные копии делаются инкрементно, то есть каждый новый снапшот содержит данные, добавленные с момента последнего снимка. Для восстановления снапшотов Elasticsearch — как всего снапшота, так и отдельных индексов — можно использовать API восстановления. В процессе восстановления можно также изменить имя и настройки индексов. Однако перед созданием снапшота необходимо зарегистрировать репозиторий.

Рассмотрим, как создать репозиторий на локальном хранилище, на примере локальной машины для создания снапшотов и восстановления.

Создание репозитория

Прежде чем делать снапшот в Elasticsearch, необходимо создать репозиторий для хранения файлов снапшотов. Каждый кластер должен иметь свой отдельный репозиторий. Чтобы создать репозиторий на локальной машине, выполните следующие действия:

1. Выберите каталог, в котором будут храниться файлы снапшотов. Мы используем `"/var/tmp/backups"`, но вы можете создать каталог резервных копий в любом месте с любым именем.
2. Установите соответствующие разрешения для каталога резервных копий, чтобы Elasticsearch мог записывать снапшоты в это место. Используйте следующую команду, чтобы предоставить Elasticsearch разрешения на владение:

```
chown -R elasticsearch /var/tmp/backups
```

3. Добавьте путь к конфигурации репозитория в файл `elasticsearch.yml`. Откройте файл `elasticsearch.yml` и добавьте строку:

```
path.repo: ["/var/tmp/backups"]
```

4. Настроив путь к репозиторию, можно создать его с помощью API репозитория. Для этого выполните такую команду:

```
a. PUT _snapshot/anurag_backup
b.
c. {
d.   "type": "fs",
e.   "settings": {
f.     "location": "/var/tmp/backups"
g.   }
g. }
```

5. Чтобы снять репозиторий с регистрации, выполните команду:

```
DELETE _snapshot/anurag_backup
```

6. Чтобы просмотреть подробную информацию обо всех репозиториях, выполните команду:

```
GET /_snapshot/_all
```

Эта команда выведет список всех доступных репозиториев. После того как репозиторий `"anurag_backup"` создан, можно делать снапшот.

Создание снапшота

Теперь, когда репозиторий готов, сделаем снапшот некоторых индексов, чтобы понять процесс создания снапшотов и восстановления. Можно сделать снапшот

всех индексов в кластере или указать конкретные индексы. Чтобы сделать снапшот всех индексов в кластере, выполните следующую команду:

```
PUT _snapshot/anurag_backup/snapshot_all_indices
```

Эта команда запустит процесс резервного копирования и предоставит немедленный ответ. Если вы хотите дождаться завершения снапшота перед получением ответа, можно добавить в команду флаг `wait_for_completion`. Например:

```
PUT _snapshot/anurag_backup/snapshot_all_indices?wait_for_completion=true
```

Если вы хотите сделать снапшот только определенных индексов, укажите имена индексов в формате, разделенном запятыми, используя ключ `indices`. Например:

1. PUT _snapshot/anurag_backup/snapshot_few_indices
2. {
3. "indices": "index1, index2".
4. }

Эта команда создаст снапшот для указанных индексов. Чтобы просмотреть подробности снапшота, используйте команду

```
GET /_snapshot/anurag_backup/snapshot_few_indices
```

В ответе отобразятся подробности снапшота `snapshot_few_indices`, включая имена индексов, использованные для него. Чтобы просмотреть все снапшоты в хранилище, используйте команду

```
GET /_snapshot/anurag_backup/_all
```

Эта команда выведет список всех снапшотов репозитория `anurag_backup`. Чтобы вывести все снапшоты для всех доступных репозиториев, выполните команду

```
GET /_snapshot/_all
```

Имя репозитория в запросе выше не указывается, чтобы запрос нашел и вывел список снапшотов во всех доступных репозиториях. Чтобы увидеть текущий, еще не завершенный снапшот, используйте команду

```
GET /_snapshot/anurag_backup/_current
```

Мы рассмотрели порядок создания снапшотов для одного или нескольких индексов. Чтобы удалить снапшот, выполните команду

```
DELETE /_snapshot/anurag_backup/snapshot_few_indices
```

Эта команда удалит снапшот `snapshot_few_indices`. Используя эти API снапшотов, можно создавать, просматривать и удалять снапшоты. Итак, мы

рассмотрели, как создавать хранилище и делать снапшоты. Теперь перейдем к восстановлению из снапшотов в Elasticsearch.

ВОССТАНОВЛЕНИЕ ИЗ СНАПШОТА

Для восстановления из снапшота используем конечную точку `_restore`. По умолчанию восстанавливаются все индексы из снапшота. Однако также можно выбрать конкретные индексы для восстановления. Процесс восстановления прост и требует использования конечной точки `_restore` в API. Вот пример:

```
POST /_snapshot/my_backup/snapshot_name/_restore
```

Восстановление можно выполнить на любом функционирующем кластере Elasticsearch. Если индекс, который требуется восстановить, уже активен в кластере, необходимо сначала закрыть его. Кроме того, количество шардов должно быть одинаковым. Если индекс доступен и закрыт, Elasticsearch откроет его в процессе восстановления. Если индекс не существует, Elasticsearch создаст его.

Чтобы восстановить определенные индексы из снапшота, используем следующий запрос:

```
1. POST /_snapshot/my_backup/snapshot_name/_restore
2. {
3.   "indices": "index1,index2",
4.   "ignore_unavailable": true,
5.   "include_aliases": false,
6.   "include_global_state": false
7. }
```

В этом примере мы восстановили из снапшота `index1` и `index2`. Доступны дополнительные опции, такие как `include_aliases`, которая позволяет включать псевдонимы в процесс восстановления. По умолчанию параметр `include_global_state` имеет значение `false`, это означает, что состояние кластера снапшота, включая шаблоны индексов, пайплайны ввода и постоянные настройки, не будет восстановлено.

Рассмотренные опции позволяют создавать снапшоты кластеров Elasticsearch для одного или нескольких индексов и восстанавливать их на активном кластере.

ОБЩАЯ СХЕМА ELASTIC

Elastic Common Schema (ECS) — это спецификация, разработанная для обеспечения согласованности структуры данных, хранящихся в Elasticsearch. Она

предлагает стандартизованный способ именования полей, что упрощает анализ, визуализацию и поиск данных.

Зачем нужна общая схема?

В данных, полученных из разных источников, могут встречаться общие поля с разными именами. Например, поле IP может называться по-разному в файле лога, данных метрик и данных APM. Это может создать проблемы при запросах и анализе данных, поскольку придется писать несколько запросов и использовать условия OR для обработки этих вариантов. Отсутствие согласованности усложняет задачи анализа данных и увеличивает накладные расходы.

ECS решает эти проблемы, определяя общий набор имен полей для схожих сущностей в разных источниках данных. Приняв ECS, можно сократить накладные расходы на сопоставление полей и легко идентифицировать поля, представляющие одну и ту же сущность, например IP-адреса или имена пользователей.

ECS предоставляет ряд преимуществ, в том числе:

- **Согласованность.** ECS обеспечивает хранение данных из разных источников в согласованном формате, что облегчает их анализ и визуализацию.
- **Эффективность.** ECS помогает сократить время и усилия, необходимые для индексации и поиска данных.
- **Расширяемость.** ECS разработан как расширяемая система, поэтому к данным можно добавлять пользовательские поля по мере необходимости.

Некоторые основные особенности ECS:

- **Общие имена полей.** ECS определяет набор имен полей для общих сущностей данных, таких как IP-адреса, имена пользователей и метки времени. Это облегчает поиск и анализ данных из разных источников.
- **Правильно определенные типы данных.** ECS определяет типы данных для каждого поля, что обеспечивает хранение данных в согласованном формате. Это облегчает выполнение операций с данными, такие как агрегация и соединение.
- **Документация.** ECS хорошо документирована, что облегчает ее освоение. Кроме того, доступен ряд инструментов и ресурсов, которые помогут начать работу с ECS.

Если вы ищете способ повысить согласованность, эффективность и расширяемость данных Elasticsearch, то ECS — отличный вариант, который стоит рассмотреть.

Введение в схему ECS

ECS – это спецификация с открытым исходным кодом, определяющая общий набор полей для различных типов документов. Она позволяет получать данные из различных источников и предоставляет единое соглашение об именовании общих полей. Это позволяет работать с разными наборами данных без необходимости разбираться в специфических названиях полей в каждом источнике.

С помощью ECS можно централизованно анализировать данные, поступающие из разных источников. Она способствует унификации моделирования данных, позволяя получать доступ к таким полям, как IP-адреса, используя согласованное имя поля (например, `ip`), независимо от соглашений об именовании источника (например, `remote_ip`, `user.ip` или `src_ip`).

ECS распределяет элементы данных по трем уровням в соответствии со своей таксономией:

- **Основные поля ECS.** Эти поля являются общими для разных вариантов использования и полностью определены в Elasticsearch. Они позволяют выполнять анализ данных, поиск, визуализацию и другие операции.
- **Расширенные поля ECS.** Эти поля являются расширениями основных полей ECS. Хотя они частично определены, они все равно подпадают под объекты верхнего уровня, указанные в ECS.
- **Пользовательские поля.** Эти поля определяются пользователем, а не спецификацией ECS. Они обеспечивают гибкость и могут существовать с полями ECS без конфликтов.

ECS позволяет добиться большей согласованности и совместимости в процессах моделирования и анализа данных в Elasticsearch.

Общие рекомендации ECS

При работе с ECS важно следовать определенным рекомендациям, чтобы обеспечить согласованность и совместимость. Вот некоторые общие рекомендации, которые следует учитывать:

- **Каждый документ должен содержать поле @timestamp.** Поле `@timestamp` в ECS является обязательным и должно присутствовать в каждом документе. Оно представляет собой метку времени, когда произошло событие.
- **Используйте типы данных, определенные в ECS.** Для каждого поля рекомендуется использовать типы данных, определенные в ECS. Это помогает поддерживать согласованность и обеспечивает надлежащий анализ и визуализацию данных.

- **Включите поле `ecs.version`.** В поле `ecs.version` указывается используемая версия ECS. Это поможет понять модель данных и любые будущие изменения или обновления ECS.
- **Сопоставьте максимум возможных полей с ECS.** Чтобы использовать все преимущества ECS, сопоставьте как можно больше полей с полями, определенными ECS. Это облегчает стандартизацию и обеспечивает единую структуру для различных типов документов.

Рекомендации по именованию полей ECS

В дополнение к общим рекомендациям существуют специальные рекомендации по именованию полей в ECS. Эти рекомендации обеспечивают последовательность и ясность в названиях полей.

- **Имена полей должны быть записаны в нижнем регистре.** Используйте строчные буквы в названиях полей, чтобы поддерживать согласованность во всей модели данных ECS.
- **Объединяйте слова с помощью символов подчеркивания.** Используйте символы подчеркивания (`_`) для разделения слов в имени поля. Например, `user_agent`.
- **Избегайте специальных символов, кроме подчеркивания.** В названиях полей следует избегать специальных символов, за исключением символа подчеркивания (`_`).
- **Вкладывайте поля с помощью точек.** Чтобы создать вложенные наборы полей, используйте точки (`.`) для разделения имен полей. Например, `host.ip` представляет собой поле IP-адреса в наборе полей `host`.
- **Вкладывайте наборы полей от общего к конкретному.** При вложении наборов полей следуйте иерархической структуре от общего к конкретному. Например, `host.ip` вложен в набор полей `host`.
- **Избегайте повторений.** Избегайте избыточных имен полей. Если имя поля можно представить более просто, не теряя его смысла, выбирайте более простой вариант. Например, используйте `host.ip` вместо `host.host_ip`.
- **По возможности избегайте сокращений.** Для повышения удобочитаемости и ясности рекомендуется по возможности избегать сокращений в названиях полей.
- **Используйте настоящее время для названий полей.** Если поле не представляет собой историческую информацию, используйте глаголы в настоящем времени в названиях полей. Это поможет сохранить последовательность и ясность семантики поля.

- Используйте имена в единственном или множественном числе в зависимости от ситуации.** Выбирайте имена полей, которые отражают соответствующую форму единственного или множественного числа в зависимости от типа данных, которые представляет поле.

Следуя этим рекомендациям, вы обеспечите соответствие данных схеме ECS и получите преимущества последовательной и стандартизированной структуры в Elasticsearch.

Начало работы с ECS

При работе с ECS полезно убедиться, что вводимые данные соответствуют спецификациям ECS. Это позволит централизовать и упростить анализ, поиск и визуализацию данных из разных источников. Вот как начать работу с ECS:

- Интеграция с Beats.** Если вы используете последнюю версию Beats для ввода данных, вы уже используете ECS. В Beats реализованы спецификации ECS, поэтому генерируемые ими данные соответствуют ее структуре.
- Интеграция с Logstash.** Если вы вводите данные с помощью Logstash, вы можете настроить конфигурацию Logstash, следуя спецификациям ECS. Это гарантирует, что данные, поступающие в Logstash, будут соответствовать формату ECS и это позволит централизованно управлять данными и анализировать их.
- Чтобы убедиться, что документ соответствует форматированию ECS, можно проверить наличие поля `ecs.version`. Это поле содержит номер версии ECS и подтверждает, что документ соответствует структуре ECS.

Например, рассмотрим запись в логе Apache:

```
1. 127.0.0.1 - - [09/Jul/2023:11:05:07 +0100] "GET / user HTTP/1.1" 200 2571
    "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_0) AppleWebKit/537.36
    (KHTML- ML, like Gecko) Chrome/70.0.3538.102 Safari/537.36"
```

Составим список имен полей, которые она содержит в соответствии с ECS:

1. `@timestamp`
2. `ecs.version`
3. `event.dataset`
4. `event.original`
5. `http.request.method`
6. `http.request.body.bytes`
7. `http.response.status_code`
8. `http.version`
9. `host.hostname`

```
10. message
11. service.name
12. service.type
13. service.geo.*
14. source.ip
15. url.original
16. user.name
17. user.agent.*
```

Имена полей выше определены согласно спецификации ECS, и мы также видим в списке поле `ecs.version`, в котором содержится номер версии ECS.

Мы уже говорили, что ECS помогает упростить поиск, анализ или визуализацию. Например, если требуется найти IP без ECS, поисковый запрос может быть следующим:

```
src:127.0.0.1 OR client_ip:127.0.0.1 OR apache2.access.remote_ip:127.0.0.1
OR context.user.ip:127.0.0.1 OR src_ip:127.0.0.1
```

Если мы используем ECS, тот же запрос можно упростить:

```
source.ip: 127.0.0.1
```

Точно так же можно упростить и все остальные операции, которые можно выполнять с этими данными. Поэтому важно использовать спецификацию ECS при получении данных из любого источника в Elasticsearch.

МАСШТАБИРОВАНИЕ КЛАСТЕРА ELASTICSEARCH

Масштабирование кластера Elasticsearch — важнейший вопрос управления и поддержания его производительности по мере увеличения объема данных и пользовательского трафика. Для удовлетворения растущих потребностей можно использовать две распространенные стратегии масштабирования: вертикальное и горизонтальное масштабирование.

Вертикальное масштабирование

Вертикальное масштабирование подразумевает добавление дополнительных ресурсов, таких как процессор, оперативная память или хранилище, на отдельные узлы кластера Elasticsearch. Эта стратегия направлена на увеличение мощности существующих узлов для обработки возросших рабочих нагрузок и повышения производительности. Такой подход уместен, если существующие узлы обладают достаточным резервом для эффективной работы с дополнительными ресурсами.

При реализации вертикального масштабирования важно:

- отслеживать использование ресурсов каждого узла, чтобы выявить узкие места или ограничения;
- определить конкретные ресурсы, требующие обновления, на основе наблюдаемых показателей производительности;
- тщательно планировать и выполнять обновления, чтобы свести к минимуму время простоя или нарушения работы кластера;
- проводить постоянный мониторинг производительности кластера и корректировать распределение ресурсов для поддержания оптимальной производительности.

Вертикальное масштабирование особенно полезно, когда на существующих узлах имеется свободное дисковое пространство и достаточная пропускная способность сети, а дополнительные аппаратные ресурсы могут обеспечить значительный прирост производительности, не требуя серьезных изменений в инфраструктуре.

Помните, что спецификации оборудования и процесс модернизации зависят от конкретных требований и характеристик развертывания Elasticsearch. Для выявления необходимости вертикального масштабирования и обеспечения оптимальной конфигурации кластера Elasticsearch очень важно проводить регулярный мониторинг производительности и осуществлять планирование мощностей.

Горизонтальное масштабирование

Горизонтальное масштабирование подразумевает добавление новых узлов в кластер Elasticsearch. При горизонтальном масштабировании рабочая нагрузка распределяется между несколькими узлами, что позволяет увеличить емкость хранилища данных, повысить производительность запросов и отказоустойчивость. Эта стратегия подходит, когда существующие узлы достигают предела своей мощности и не могут эффективно справляться с растущей рабочей нагрузкой.

При реализации горизонтального масштабирования важно:

- определить необходимое количество узлов, исходя из ожидаемого объема данных, пользовательского трафика и желаемого уровня производительности;
- настроить кластер на автоматическую балансировку распределения шардов между новыми узлами, чтобы равномерно распределить данные;
- обеспечить надлежащую коммуникацию и механизмы обнаружения между узлами для поддержания целостности кластера и его бесперебойной работы;

- постоянно следить за состоянием кластера и при необходимости восстанавливать баланс шардов, чтобы оптимизировать распределение данных по узлам.

Чтобы проиллюстрировать преимущества горизонтального масштабирования, предположим, что в приложении происходит внезапный всплеск пользовательского трафика в часы пик. Возросшая рабочая нагрузка эффективно распределяется между пятью узлами, предотвращая перегрузку одного узла и обеспечивая бесперебойную работу даже в периоды высокого спроса. Кроме того, горизонтальное масштабирование повышает отказоустойчивость, поскольку кластер может продолжать функционировать даже в том случае, если один или несколько узлов выходят из строя или находятся на обслуживании.

Стратегии масштабирования можно комбинировать в зависимости от конкретных требований кластера Elasticsearch. Например, вертикальное масштабирование может быть использовано для оптимизации производительности отдельных узлов, а горизонтальное — для обеспечения дополнительной емкости и отказоустойчивости.

Чтобы определить потребность в масштабировании, необходимо проводить регулярный мониторинг производительности кластера, использования ресурсов и моделей роста. Кроме того, рекомендуется тестировать и проверять стратегии масштабирования в контролируемой среде, прежде чем внедрять их на продакшен.

Благодаря использованию соответствующих стратегий масштабирования кластеры Elasticsearch эффективно справляются с растущими рабочими нагрузками, обеспечивая масштабируемость и поддерживая оптимальную производительность и доступность по мере роста кластера.

МОНИТОРИНГ ELASTICSEARCH

Для обеспечения работоспособности, производительности и доступности кластера Elasticsearch очень важно проводить его мониторинг. Отслеживая различные метрики и показатели, можно заблаговременно выявлять проблемы, оптимизировать производительность и принимать обоснованные решения о планировании мощностей и распределении ресурсов. Вот на что следует обращать внимание при проведении мониторинга Elasticsearch:

- **Состояние кластера.** Следите за общим состоянием кластера Elasticsearch, регулярно проверяя API состояния (`/_cluster/health`). Этот API предоставляет информацию о состоянии кластера, включая количество узлов, активных и неактивных шардов, а также общий статус кластера (зеленый, желтый или красный).

- **Метрики узлов.** Отслеживайте отдельные узлы Elasticsearch, чтобы получить информацию об использовании их ресурсов, производительности и статусе эксплуатации. Важными метриками мониторинга являются использование процессора, потребление памяти, дисковый ввод-вывод, сетевой трафик и сборка мусора. Доступ к этим метрикам можно получить через API статистики узла (`/_nodes/stats`) или с помощью специальных инструментов мониторинга, например Elastic Stack.
- **Производительность индексирования и поиска.** Отслеживайте производительность индексирования и поиска, чтобы обеспечить эффективный ввод и извлечение данных. Следите за такими метриками, как скорость индексирования, задержка поиска, пропускная способность запросов и коэффициент попадания в кэш. Анализ этих метрик поможет выявить узкие места, оптимизировать производительность запросов и при необходимости скорректировать стратегии индексирования.
- **Распределение кластеров и шардов.** Контролируйте распределение кластеров и шардов, чтобы обеспечить распределение данных и балансировку нагрузки между узлами. Отслеживайте такие метрики, как распределение шардов, количество первичных шардов и реплик и баланс распределения шардов. Неравномерное распределение шардов или неназначенные шарды могут указывать на проблемы состояния кластера или несбалансированное использование ресурсов.
- **Управление использованием диска и пространства.** Контролируйте использование диска, чтобы предотвратить нехватку места для хранения и обеспечить правильное управление индексами. Отслеживайте использование диска по узлам и индексам и настраивайте оповещения, когда объем дискового пространства достигает критического уровня. Регулярно выполняйте задачи по обслуживанию индексов, такие как сжатие индекса или удаление устаревших данных, чтобы освободить место на диске.
- **Производительность JVM.** Отслеживайте производительность виртуальной машины Java (JVM) на узлах Elasticsearch. Следите за использованием кучи JVM, поведением сборки мусора и такими метриками JVM, как распределение памяти, количество потоков и паузы JVM. Правильная настройка JVM и управление памятью могут существенно повлиять на общую производительность и стабильность Elasticsearch.
- **События и логи кластера.** Отслеживайте события и логи кластера, чтобы выявить потенциальные проблемы или аномалии. Логи Elasticsearch содержат полезную информацию об операциях кластера, ошибках индексирования, сбоях запросов и других важных событиях. Централизуйте и анализируйте логи с помощью таких инструментов, как Elastic Stack, чтобы получать информацию и эффективно устранять неполадки.

- **Оповещения и уведомления.** Настройте проактивные оповещения и уведомления, чтобы оперативно реагировать на критические проблемы или аномальное поведение. Настройте оповещения на основе определенных пороговых значений или аномалий в метриках, таких как высокое использование ЦП, низкий объем дискового пространства или сбои в работе узлов. Интеграция мониторинга Elasticsearch с системами оповещения или платформами управления инцидентами обеспечивает своевременное реагирование и сокращает время простоя.
- **Исторические данные и анализ тенденций.** Сохраняйте исторические данные мониторинга, чтобы анализировать тенденции, отслеживать изменения производительности с течением времени и выявлять долгосрочные закономерности. Хранение и анализ исторических данных может помочь в планировании мощности, выявлении снижения производительности и обнаружении аномалий и нетипичных значений.

Для Elasticsearch доступны различные решения для мониторинга, в том числе Elastic Stack (Elasticsearch, Logstash, Kibana и Beats), который предоставляет комплексную платформу для мониторинга и наблюдения. Кроме того, для мониторинга кластеров Elasticsearch можно использовать сторонние инструменты и сервисы мониторинга.

Регулярный просмотр данных мониторинга, анализ производительности и применение проактивных мер на основе полученных в ходе мониторинга сведений помогут оптимизировать производительность, стабильность и масштабируемость развертывания Elasticsearch.

ЗАКЛЮЧЕНИЕ

В этой главе мы подробно рассмотрели вопросы администрирования кластера Elasticsearch, выделив важные темы. Мы начали с изучения реализации мер безопасности в кластере Elasticsearch. Мы научились применять такие функции безопасности, как шифрование TLS и управление доступом на основе ролей (RBAC), которые обеспечивают безопасную коммуникацию и управление доступом в кластере.

Далее мы перешли к концепции псевдонимов индексов. Мы узнали, как псевдонимы обеспечивают удобный способ ссылки на индекс со вторым именем, что позволяет выполнять запросы к индексу с использованием псевдонима. Мы изучили преимущества использования псевдонимов, такие как возможность переиндексации с нулевым временем простоя и облегчение коллективного мониторинга нескольких индексов.

Еще один важный рассмотренный вопрос — создание снапшотов и восстановление из них в Elasticsearch. Мы узнали, что снапшоты служат резервными копиями кластера, позволяя восстановить данные в случае сбоев или потери данных. Мы изучили процесс создания репозиториев для хранения снапшотов и выполнения команд для инициирования создания и восстановления снапшотов. Понимание возможностей создания и восстановления снапшотов дает уверенность в безопасности и отказоустойчивости данных Elasticsearch.

Мы также рассмотрели ECS — спецификацию, которая обеспечивает согласованную структуру данных в Elasticsearch. Мы обсудили важность ECS для упрощения унифицированного анализа, поиска и визуализации данных путем предоставления общего набора имен полей и стандартов моделирования данных. Мы рассмотрели основные поля ECS, расширенные поля и возможность использования пользовательских полей при наличии особых требований к данным.

Кроме того, мы затронули важнейшую тему — мониторинг кластера Elasticsearch. Мониторинг играет важную роль в обеспечении надлежащего состояния, производительности и стабильности кластера. Отслеживая различные метрики и показатели, можно заблаговременно выявлять любые потенциальные проблемы, оптимизировать использование ресурсов и принимать обоснованные решения для повышения общей производительности кластера.

Рассмотрев вопросы администрирования, включая внедрение безопасности, псевдонимы индексов, снапшоты, Elastic Common Schema и мониторинг Elasticsearch, мы получили необходимые знания для эффективного управления и оптимизации кластера Elasticsearch.

ВОПРОСЫ

1. Какие шаги необходимо предпринять для реализации мер безопасности в кластере Elasticsearch?
2. Как создать псевдоним индекса в Elasticsearch и каковы преимущества использования псевдонимов?
3. Опишите процесс создания репозитория в Elasticsearch для снятия снапшотов.
4. Как инициировать создание снапшота и выполнить процесс восстановления в Elasticsearch?
5. Каковы стратегии масштабирования кластера Elasticsearch?

Анураг Шривастава

Elasticsearch для разработчиков:
индексирование, анализ, поиск и агрегирование данных

2-е издание

Перевел с английского А. Ларин

Научный редактор А. Белых

Руководитель дивизиона

Ю. Сергиенко

Руководитель проекта

А. Питиримов

Ведущий редактор

Е. Строганова

Литературный редактор

М. Трусковская

Художественный редактор

В. Мостицан

Корректоры

С. Беляева, Н. Викторова

Верстка

Л. Егорова

Изготовлено в России. Изготовитель: ООО «Прогресс книга».

Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,

Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 05.2025. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные
профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 25.03.25. Формат 70×100/16. Бумага офсетная. Усл. п. л. 27,090. Тираж 500. Заказ 0000.

КРОК

СОЗДАЕМ НАСТОЯЩЕЕ,
ИНТЕГРИРУЕМ БУДУЩЕЕ



croc.ru

КРОК – технологический партнер с комплексной экспертизой в области построения и развития инфраструктуры, внедрения информационных систем, разработки программных решений и сервисной поддержки.

Центры компетенций КРОК фокусируются на ключевых отраслевых кластерах — промышленность, финансовый сектор, розничные продажи, муниципальное управление, спорт и культура.

Ежегодно сотни проектов КРОК становятся системообразующими для экономики и социально-культурной сферы.

