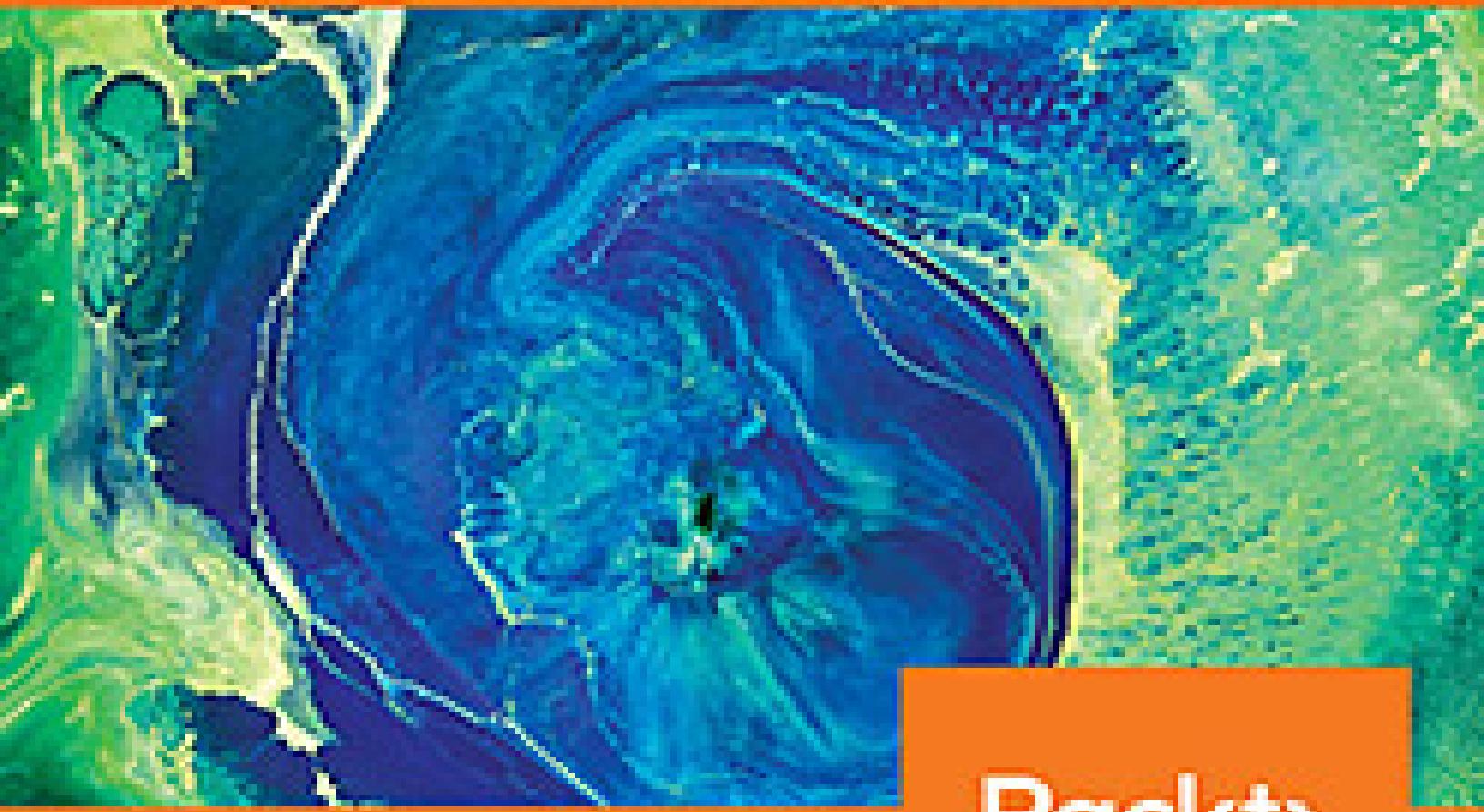


Elasticsearch Kibana Logstash

и подобные системы нового поколения

Пранав Шукла, Шарат Кумар



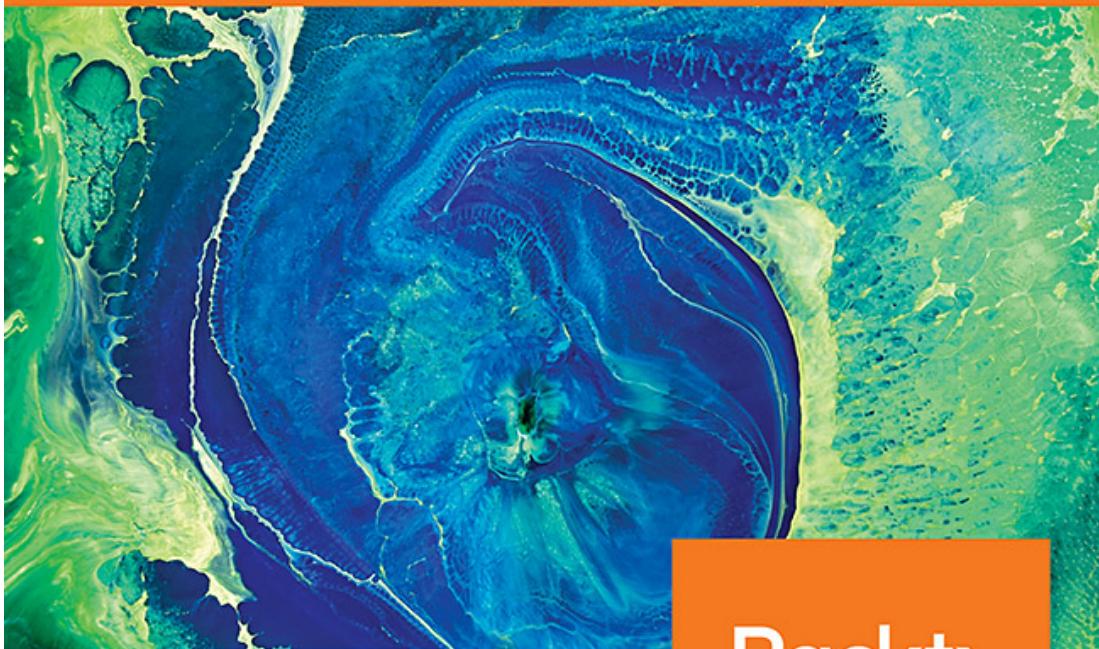
Сигма

Packt

Elasticsearch Kibana Logstash

И ПОИСКОВЫЕ СИСТЕМЫ НОВОГО ПОКОЛЕНИЯ

Пранав Шукла, Шарат Кумар



Packt

ПИТЕР®

Пранав Шукла, Шарат Кумар

Elasticsearch, Kibana, Logstash и поисковые системы нового поколения



2019

Переводчики *С. Белов , Е. Сандицкая*
Технический редактор *Н. Гринчик*
Литературный редактор *Н. Гринчик*
Художники *Н. Гринчик, С. Заматевская , Г. Синякина
(Маклакова)*
Корректоры *О. Андриевич, Г. Блинов, Е. Павлович*
Верстка *Г. Блинов*

Пранав Шукла, Шарат Кумар
Elasticsearch, Kibana, Logstash и поисковые системы нового поколения . — СПб.: Питер, 2019.

ISBN 978-5-4461-1024-7
© [ООО Издательство "Питер"](#), 2019

Все права защищены. Никакая часть данной книги не может быть воспроизведена в какой бы то ни было форме без письменного разрешения владельцев авторских прав.

Об авторах

Пранав Шукла — основатель и руководитель компании Valens DataLabs, инженер, муж и отец двоих детей. Разработчик архитектур больших данных и профессиональный программист, использующий языки программирования на базе JVM. Пранав более 14 лет занимается разработкой корпоративных приложений для компаний и стартапов Fortune 500. Его основная специализация — создание масштабируемых, управляемых данными приложений на основе JVM, Java/Scala, экосистемы Hadoop, Apache Spark и баз данных NoSQL. Активно развивается в сферах, связанных с организацией больших данных, аналитикой и машинным обучением.

Пранав основал Valens DataLabs, чтобы помочь другим компаниям использовать данные для повышения своей конкурентоспособности. Valens DataLabs специализируется на создании облачных приложений нового поколения для работы с большими данными и веб-технологиями. Работа компании основана на использовании гибких практик, принципов бережливого производства, разработки на основе тестов и поведения, постоянной интеграции и непрерывного развертывания устойчивых программных систем.

В свое свободное время Пранав любит читать, играть на музыкальных инструментах, петь, слушать музыку и смотреть матчи по крикету. Вы можете связаться с ним по электронной почте: pranav.shukla@valensdatalabs.com — и подписаться на него в «Твиттере»: [@pranavshukla81](https://twitter.com/pranavshukla81).

Я хотел бы поблагодарить свою жену Крутини Шукла за ее безоговорочную любовь и поддержку, наших сыновей Саухадра и Пратишха, моих родителей Шарада Шукла и Варшу Шукла.

Благодарю также своего брата Вишала Шукла за вдохновение и особую роль в моей карьере и написании этой книги. Отдельные благодарности для Партха Мистри, Гопала Хангара и Кришны Мита за их ценные отклики о книге. И спасибо остальным, кто привнес посильный вклад в мою карьеру: Умешу Каккаду, Эдди Мужену, Варту Франсену, Правину Саменени, Виноду Пателу, Гопаху Шаху и Сачину Бакши.

Шарат Кумар М.Н. получил степень магистра по компьютерным наукам в Техасском университете, Даллас, США. Он уже более десяти лет работает в ИТ-индустрии, в данный момент занимает должность разработчика решений на Oracle для Elasticsearch, является приверженцем Elastic Stack. Будучи заядлым оратором, он выступал на нескольких научно-технических конференциях, в том числе Oracle Code Event. Шарат — аттестованный преподаватель Elastic (Elastic Certified Instructor) — один из нескольких технических экспертов в мире, кому компания Elastic Inc. предоставила официальное право проводить тренинги «от создателей Elastic». Он также является энтузиастом в машинном обучении и науке о данных.

В свободное время Шарат занимается горным туризмом, любит слушать музыку, играть со своими питомцами Гудду и Мило. Он также расширяет свои навыки программирования на Python для анализа биржевых рынков. Вы можете связаться с ним по электронной почте: mnsk07@gmail.com.

Я хотел бы поблагодарить своих родителей, Гита и Нанжайю, сестру Шилпу М.Н., зятя Сридхара и моих друзей — без их поддержки я бы не смог закончить свою часть книги вовремя. Также говорю спасибо сотрудникам издательства Packt Publishing

*(особенно Черил, Сэмьюэлю, Варше, Сагару) за предоставление
отличной возможности поучаствовать в этом захватывающем
путешествии.*

О научном редакторе

Марцело Очоа работает в лаборатории факультета точных наук в Национальном университете Центрального Буэнос-Айреса (Universidad Nacional del Centro de la Provincia de Buenos Aires), Аргентина. Он является техническим директором компании Scotas (www.scotas.com), которая специализируется на решениях в псевдореальном времени с использованием технологий Apache Solr и Oracle. Марцело успевает работать в университете и заниматься проектами, связанными с Oracle и технологиями больших данных. Ранее он работал с базами данных, веб- и Java-технологиями. В мире XML Марцело известен как разработчик DB Generator для проекта Apache Cocoon. Он принимал участие в создании таких проектов с открытым исходным кодом, как DBPrism, DBPrism CMS, а также Restlet.org, где работал над Oracle XDB Restlet Adapter, который является альтернативой для записи нативных веб-сервисов REST внутри базы данных JVM.

С 2006 года он принимает участие в программе Oracle ACE, а в последнее время подключился к работе над проектом Docker Mentor.

Марцело является соавтором таких книг, как *Oracle Database Programming Using Java and Web Services* от издательства Digital Press и *Professional XML Databases* от Wrox Press. Он был техническим обозревателем в некоторых книгах издательства Packt: *Mastering Elastic Stack*, *Mastering Elasticsearch 5.x, Third Edition*, *Elasticsearch 5.x Cookbook, Third Edition* и пр.

Предисловие

Elastic Stack — это мощный набор инструментов для распределенного поиска, анализа, сбора и визуализации данных в массивах данных различного размера. Новейший Elastic Stack 6.0 отличается новыми функциями и возможностями, которые позволяют пользователям находить уникальные решения. В этой книге мы дадим вам фундаментальное понимание работы программного обеспечения и научим эффективно пользоваться мощными приложениями по обработке данных в реальном времени.

Сначала мы кратко рассмотрим новые функции Elastic Stack 6.0, после чего вы узнаете, как устанавливать инструменты, и познакомитесь с их базовыми конфигурациями. Далее вы увидите, как использовать Elasticsearch для распределенного поиска и анализа, Logstash — для сбора данных, а Kibana — для визуализации данных. Будет продемонстрировано создание пользовательских плагинов с помощью Kibana и Beats. Поговорим об Elastic X-Pack — полезном расширении для реальной защиты и мониторинга. Мы также дадим вам полезные рекомендации по использованию Elastic Cloud и развертыванию Elastic Stack в производственной среде.

Структура книги

Глава 1 «Введение в Elastic Stack» знакомит с основными компонентами Elastic Stack, объясняет их роль в общей структуре, описывает назначение каждого компонента. В этой главе также рассказывается о необходимости распределенного, масштабируемого поиска и анализа, которые достигаются с помощью Elasticsearch. В конце приводится руководство по

скачиванию и установке Elasticsearch и Kibana, чтобы можно было начать работу с этими инструментами.

Глава 2 «Начало работы с Elasticsearch» знакомит с ключевыми принципами работы поисковой системы Elasticsearch, которая является основой Elastic Stack. Вы познакомитесь с такими концепциями, как индексы, типы, узлы и кластеры. Вы также узнаете, как использовать REST API для выполнения основных операций.

Глава 3 «Поиск — вот что важно» фокусируется на методах поиска, предоставляемых Elasticsearch. Вы узнаете об основах анализа текста, токенизаторах, анализаторах, особенностях релевантного поиска. В этой главе также приводятся практические примеры релевантного поиска.

Глава 4 «Анализ данных с помощью Elasticsearch» расскажет о различных типах агрегации. Она включает примеры, которые позволяют вам лучше понять принципы анализа данных. Вы научитесь использовать различные виды агрегаций — от простейших до сложных, чтобы сориентироваться в огромных массивах данных. Прочитав эту главу, вы узнаете, когда и какой вариант агрегации лучше задействовать.

Глава 5 «Анализ журнальных данных» содержит информацию о необходимости использования Logstash, его архитектуре, установке и настройке. В Elastic 5 предоставляется инструмент Ingest Node, который может заменить конфигурацию Logstash. Прочитав эту главу, вы научитесь создавать контейнеры с использованием Elastic Ingest Node.

Глава 6 «Разработка контейнеров с помощью Logstash» дает основополагающие знания о Logstash, позволяющем динамически идентифицировать данные из различных источников и нормализовать их с помощью выбранных фильтров. Вы узнаете, как наличие широкого набора фильтров ставит Logstash в один ряд с другими фреймворками потоковой

обработки в реальном и околореальном времени без написания кода. Вы также познакомитесь с платформой Beats и компонентом FileBeat, используемым для транспортировки лог-файлов (файлов регистрации) с удаленных машин.

Глава 7 «Визуализация данных в Kibana» показывает эффективность использования Kibana для визуализации и впечатляющего представления ваших данных. На примере простого набора данных описывается создание визуализаций в пару кликов.

Глава 8 «Elastic X-Pack» рассказывает о расширении Elasticsearch. К этому времени вы уже изучите Elasticsearch и его ключевые компоненты для создания контейнеров данных и сможете подключать расширения для решения конкретных задач. В этой главе вы прочтете, как установить и настроить компоненты X-Pack в Elastic Stack, усвоите азы безопасности и мониторинга и научитесь добавлять различные уведомления.

Глава 9 «Запуск Elastic Stack в работу» дает рекомендации по запуску комплекса Elastic Stack в промышленную эксплуатацию. Вы получите рекомендации по внедрению вашего приложения в работу и изменению стандартных настроек согласно требованиям эксплуатации. Вы также узнаете, как использовать облачные сервисы Elastic Cloud.

Глава 10 «Создание приложения для анализа данных с датчиков» описывает создание приложения для анализа и обработки данных, получаемых из различных источников. Вы узнаете, как моделировать данные в Elasticsearch, создавать контейнеры данных и визуализировать их в Kibana. Вы также научитесь эффективно использовать компоненты X-Pack для обеспечения безопасности и мониторинга ваших контейнеров, получать уведомления о различных событиях.

Глава 11 «Мониторинг серверной инфраструктуры» демонстрирует возможности использования Elastic Stack для

настройки мониторинга в реальном режиме времени для серверов и приложений, которые созданы целиком на Elastic Stack. Вы познакомитесь еще с одним компонентом платформы Beats — Metricbeat, который применяется для мониторинга серверов/приложений.

Что вам понадобится для работы

Все примеры, которые приводятся в книге, созданы с использованием следующих версий инструментов:

- Elasticsearch 6.0;
- Kibana 6.0.

Для кого предназначена книга

Книга предназначена для специалистов, работающих с большими объемами данных и желающих надежно извлекать их из любого источника в любом формате, а также искать, анализировать и визуализировать данные в режиме реального времени. Эта книга для вас, если вам необходимо фундаментальное понимание работы Elastic Stack в сферах распределенных вычислений и обработки данных в реальном времени. Знание JSON будет преимуществом, но не является обязательным. Также не требуется какой-либо опыт работы с другими версиями Elastic Stack.

Условные обозначения

В книге вы увидите текст, оформленный различными стилями. Ниже приводятся примеры стилей и объясняется, что означает это форматирование.

Фрагменты кода в тексте, названия таблиц баз данных, названия папок, имена и расширения файлов, пользовательский ввод выделены моношириным шрифтом. Адреса URL и ссылки на Twitter — **рубленым**. Например: «Следующий код читает ссылки и присваивает их функции BeautifulSoup». Фрагмент кода выглядит таким образом:

```
#import packages into the project
from bs4 import BeautifulSoup
from urllib.request import urlopen
import pandas as pd
```

Когда требуется обратить особое внимание на конкретные фрагменты кода, они выделяются жирным шрифтом:

```
[default] exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail(u100)
exten => s,102,Voicemail(b100)
exten => i,1,Voicemail(s0)
```

Любой ввод или вывод командной строки отображается следующим образом:

```
C:\Python34\Scripts> pip install -upgrade pip
C:\Python34\Scripts> pip install pandas
```

Новые термины или *важные слова* выделены курсивом. Слова, которые вы видите на экране, например в меню или диалоговых окнах, в тексте выглядят следующим образом: «Для скачивания новых модулей перейдите по адресу **Файл—>Настройки—>Имя проекта—>Интерпретатор проекта**».



Предупреждения или важные заметки выглядят так.



Советы выглядят так.

Загрузка примеров кода

Примеры кода для выполнения упражнений из этой книги доступны для скачивания по адресу <https://github.com/PacktPublishing/Learning-Elastic-Stack-6>. Чтобы скачать файлы, выполните следующие действия.

1. Перейдите по адресу <https://github.com/PacktPublishing/Learning-Elastic-Stack-6>.
2. Нажмите кнопку **Clone or download** (Клонировать или скачать).
3. На открывшейся панели щелкните на ссылке **Download Zip** (Скачать Zip).

После того как архив будет загружен, можете распаковать его в нужную папку, используя последнюю версию одной из нижеперечисленных программ:

- WinRAR или 7-Zip для Windows;
- Zipeg, iZip или UnRarX для macOS;
- 7-Zip или PeaZip для Linux.

Скачивание цветных изображений, использованных в книге

Мы предоставляем оригинальный PDF-файл с цветными изображениями, скриншотами и схемами, которые используются в этой книге. На цветных иллюстрациях лучше видны изменения вывода. Вы можете скачать файл по ссылке: https://www.packtpub.com/sites/default/files/downloads/LearningElasticStack6_ColorImages.pdf.

1. Введение в Elastic Stack

Мы живем в информационную эпоху. За последние годы Интернет, мобильные приложения, социальные сети, блоги и фотосервисы произвели огромное количество данных. Эти новые источники информации невозможно обработать с помощью традиционных технологий хранения данных, стандартных реляционных баз данных. Для разработчиков приложений или бизнес-аналитиков важно сделать так, чтобы все требования по поиску и анализу в приложении были удовлетворены.

За последние несколько лет появились различные системы для хранения и обработки больших массивов данных. Среди них можно выделить проекты экосистемы Hadoop, некоторые базы данных (БД) NoSQL, а также поисковые и аналитические системы наподобие Elasticsearch. Hadoop и любая база данных NoSQL имеют свои преимущества и области применения.

Elastic Stack — обширная экосистема компонентов, которые служат для поиска и обработки данных. Основные компоненты Elastic Stack — это Kibana, Logstash, Beats, X-Pack и Elasticsearch. Ядром Elastic Stack выступает поисковая система Elasticsearch, которая предоставляет возможности для хранения, поиска и обработки данных. Утилита Kibana, которую также называют окном в Elastic Stack, является отличным средством визуализации и пользовательским интерфейсом для Elastic Stack. Компоненты Logstash и Beats позволяют передавать данные в Elastic Stack. X-Pack предоставляет мощный функционал: можно настраивать мониторинг, добавлять различные уведомления, устанавливать параметры безопасности для подготовки вашей системы к эксплуатации. Поскольку Elasticsearch является ядром Elastic Stack, мы

рассмотрим систему изнутри, начиная с самого центра и заканчивая пограничными компонентами.

В этой главе мы рассмотрим следующие темы.

- Что такое система Elasticsearch и чем она хороша.
- Краткая история Elasticsearch и Apache Lucene.
- Компоненты Elastic Stack.
- Примеры использования Elastic Stack.

Мы разберем, что собой представляет Elasticsearch и почему вам следует рассмотреть именно ее в качестве хранилища данных. Обсудив ключевые преимущества Elasticsearch, мы рассмотрим историю ее создания и технологию Apache Lucene, которая лежит в основе этой поисковой системы. Затем мы разберем несколько примеров использования Elastic Stack и подробно изучим все его компоненты.

Что такое система Elasticsearch и чем она хороша

Если вы читаете эту книгу, то наверняка уже знаете, что такое Elasticsearch. Для полноты описания приведем ее определение.

Elasticsearch — высокомасштабируемая распределенная поисковая система полнотекстового поиска и анализа данных, работающая в режиме реального времени. Утилита позволяет хранить, искать и анализировать большие объемы данных. Обычно используется в качестве базового механизма/технологии, помогая приложениям со сложными функциями поиска. Elasticsearch представляет собой основной компонент Elastic Stack.

Elasticsearch как сердце Elastic Stack играет основную роль в поиске и анализе данных. Она построена на уникальной технологии — Apache Lucene. Благодаря этому Elasticsearch в корне отличается от традиционных решений для реляционных баз данных или NoSQL. Ниже перечислены основные преимущества использования Elasticsearch в качестве хранилища данных:

- неструктурированность, документоориентированность;
- возможность поиска;
- возможность анализа данных;
- поддержка пользовательских библиотек и REST API;
- легкое управление и масштабирование;
- работа в псевдореальном времени;
- высокая скорость работы;
- устойчивость к ошибкам и сбоям.

Рассмотрим все преимущества по отдельности.

Неструктурированность и документоориентированность

Elasticsearch не навязывает четкую структуру для ваших данных; вы можете хранить любые документы JSON, которые прекрасно подходят для Elasticsearch в отличие от строк и столбцов в реляционных базах данных. Такой документ можно сравнить с записью в таблице БД. В традиционных реляционных базах данных таблицы структурированы: имеют

фиксированное количество столбцов, у каждого свой тип и размер. Но данные могут динамично меняться, что потребует поддержки новых или динамических столбцов. Документы JSON изначально поддерживают такой тип данных. Например, взгляните на следующий документ:

```
{  
  "name": "John Smith",  
  "address": "121 John Street, NY, 10010",  
  "age": 40  
}
```

Так может выглядеть запись о клиенте. В ней указаны *имя*, *адрес* и *возраст* клиента. А другая запись может выглядеть так:

```
{  
  "name": "John Doe",  
  "age": 38,  
  "email": "john.doe@company.org"  
}
```

Обратите внимание, что у второго клиента нет поля для *адреса*, но вместо него есть поле для *электронной почты*. А у остальных клиентов может быть совсем иной набор полей. Следовательно, необходима большая гибкость в отношении того, что именно может храниться в таблице.

Поиск

Основное достоинство системы Elasticsearch — в ее возможностях обработки текста. Elasticsearch отлично подходит для поиска, особенно полнотекстового. Разберемся, что это такое.



Полнотекстовый поиск представляет собой поиск по всем выражениям во всех документах, доступных в базе данных. Для этого необходимо, чтобы все содержимое всех документов было проанализировано и сохранено заранее. Когда вы слышите фразу «полнотекстовый поиск», вы наверняка представляете Google. Вы можете ввести любое выражение, и система Google начнет искать его по всем страницам в Интернете, чтобы выдать наилучшее совпадение. Простые SQL-запросы действуют иначе: они работают со столбцами типа *string* в реляционных базах данных. Обычный SQL-запрос с условием WHERE и знаком «равно» (=) или условием LIKE пытается найти точное или универсальное совпадение с данными. Запросы SQL могут в лучшем случае найти совпадение условия поиска и подстроки в пределах текста столбца.

Если вам необходим поиск, как в Google, идеально подойдет Elasticsearch. Вы можете индексировать сообщения электронной почты, текстовые документы, файлы PDF, веб-страницы или любые другие неструктурированные текстовые документы, используя ключевые слова.

Elasticsearch разбивает текстовые данные на ключевые слова, и каждое слово доступно для поиска. Для этого используются индексы Lucene. Вы можете настроить в своем приложении Google-подобный поиск, и он будет очень быстрым и гибким.

Помимо текстовых, Elasticsearch поддерживает другие типы данных: числа, даты, геолокацию, IP-адреса и др. Более

детально эти функции мы рассмотрим в главе 3.

Анализ данных

У Elasticsearch есть еще одно важное *функциональное* преимущество — возможность аналитической обработки данных. Да, то, что изначально известно как полнотекстовый движок поиска, теперь используется и как механизм для аналитики. Многие организации задействуют в работе решения для анализа данных на базе Elasticsearch.

Сегодня, с ростом количества данных, выполнить поиск означает найти иголку в стоге сена. При этом поиск дает возможность выявить нужное среди огромных массивов информации. Аналитика же, наоборот, позволяет увидеть цельную картину. Например, вы хотите узнать, сколько на вашем сайте посетителей конкретно из США в сравнении с количеством пользователей из других стран, или, возможно, вам необходимо знать, какая часть посетителей пользуется macOS, Windows или Linux.

Elasticsearch предлагает широкий спектр агрегаторов для аналитики. Системы агрегации Elasticsearch могут применяться для различных типов данных. Более подробно о возможностях анализа данных в Elasticsearch мы поговорим в главе 4.

Поддержка пользовательских библиотек и REST API

Elasticsearch имеет широчайшую поддержку пользовательских библиотек, обеспечивающую полную совместимость со многими языками программирования. Клиентские библиотеки доступны для Java, C#, Python, JavaScript, PHP, Perl, Ruby и др. Помимо официальных, существуют и неофициальные библиотеки для более чем 20 языков программирования.

Есть также многофункциональный *REST API* (*Representational State Transfer*), работающий на протоколе HTTP. REST API имеет отличную документацию и отличается хорошей гибкостью, позволяет выполнять все операции через HTTP.

Благодаря всему перечисленному Elasticsearch легко интегрировать в любое приложение для различных задач поиска или аналитики.

Легкое управление и масштабирование

Elasticsearch можно запустить на одном узле и легко масштабировать до сотен узлов. Начать работу с экземпляром узла Elasticsearch очень просто — его установочная версия не требует каких-либо изменений в конфигурации. При этом вы можете масштабировать его.



Горизонтальное масштабирование — это возможность горизонтально масштабировать систему путем запуска нескольких однотипных процессов, вместо того чтобы наращивать мощность одного процесса. Вертикальное масштабирование — это наращивание мощности одного процесса путем добавления мощности (увеличения количества ядер ЦП), памяти или дискового пространства. Бесконечное вертикальное масштабирование невозможно по финансовым и другим причинам, например, из-за недоступности более мощного аппаратного обеспечения.

В отличие от большинства традиционных баз данных, которые поддерживают только вертикальное масштабирование,

Elasticsearch может быть масштабирована горизонтально. Она может быть запущена на десятках или сотнях узлов вместо одного очень дорогого сервера. Добавить к имеющемуся кластеру Elasticsearch еще один узел так же легко, как и добавить узел в той же сети, практически без дополнительных настроек. Нет необходимости изменять клиентское приложение в зависимости от того, работает оно на одном узле или на кластере из сотен узлов.

Работа в псевдореальном времени

Обычно данные доступны для запросов в пределах секунды после того, как были проиндексированы (сохранены). Не все системы хранения больших данных имеют возможности работы в псевдореальном времени. Elasticsearch позволяет вам индексировать от тысяч до сотен тысяч документов в секунду. И они доступны для поиска почти мгновенно.

Высокая скорость работы

Поисковая система Elasticsearch использует Apache Lucene как базовую технологию. По умолчанию Elasticsearch индексирует все поля в ваших документах. Это очень важно — так вы сможете делать запросы по любым полям в ваших записях. Вам никогда не придет в голову мысль: «Все же стоило создать для этого поля индекс». Разработчики Elasticsearch настроили работу Apache Lucene для максимальной производительности и провели другие виды оптимизации, чтобы повысить скорость работы.

Устойчивость к ошибкам и сбоям

Кластеры Elasticsearch продолжают работать, даже если возникают аппаратные ошибки типа сбоя узла или неполадок сети. В случае сбоя узла все его данные копируются на другой узел в кластере. При неполадках сети Elasticsearch незаметно для пользователя выбирает ведущие копии, чтобы кластер продолжал работать. При любых неполадках узлов или сети вы можете быть уверены, что ваши данные в безопасности.

Обзор компонентов Elastic Stack

Компоненты Elastic Stack представлены на рис. 1.1. Нет необходимости использовать сразу все из них в вашем решении. Некоторые компоненты универсальны, их можно применять без Elastic Stack или других инструментов.

Взглянем на предназначение каждого компонента и на то, какое место они занимают в общем наборе (рис. 1.1).

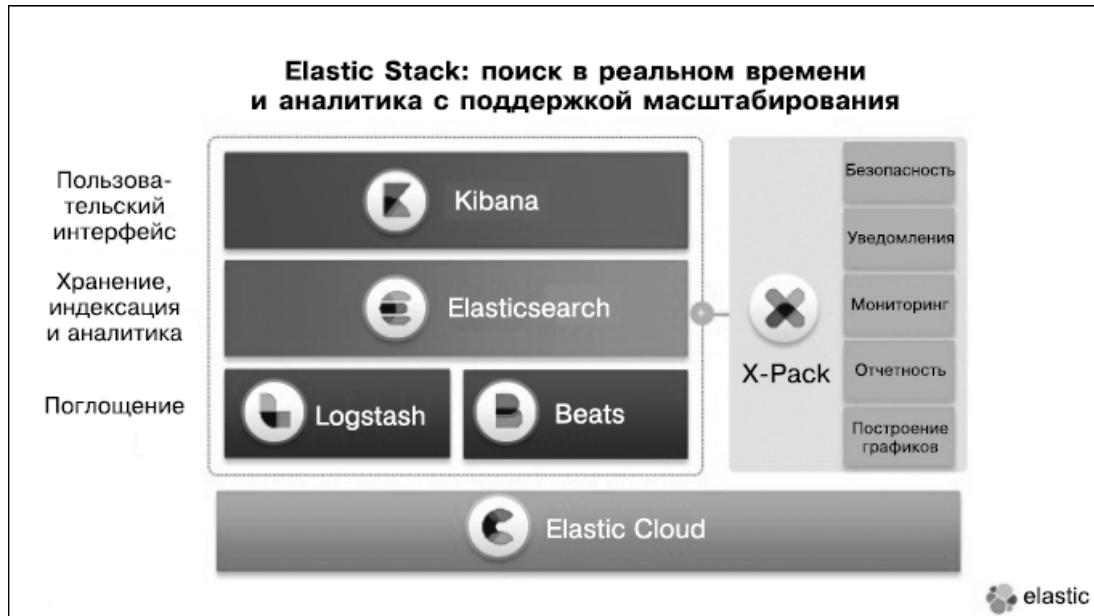


Рис. 1.1

Elasticsearch

Elasticsearch хранит все ваши данные, предоставляет возможности поиска и анализа в масштабируемом виде. Мы уже рассматривали преимущества и причины использования Elasticsearch. Вы можете работать с Elasticsearch без каких-либо других компонентов, чтобы оснастить свое приложение инструментами для поиска и анализа данных. Мы подробно рассмотрим этот компонент в главах 2–4.

Logstash

Утилита Logstash помогает централизовать данные, связанные с событиями, такие как сведения из файлов регистрации (логов), разные показатели (метрики) или любые другие данные в любом формате. Она может выполнить обработку данных до того, как сформировать нужную вам выборку. Это ключевой компонент Elastic Stack, который используется для сбора и обработки ваших контейнеров данных.

Logstash — компонент на стороне сервера. Его цель — выполнить сбор данных из обширного количества источников ввода в масштабируемом виде, обработать информацию и отправить ее по месту назначения. По умолчанию преобразованная информация поступает в Elasticsearch, но вы можете выбрать один из многих других вариантов вывода. Архитектура Logstash основана на плагинах и легко расширяется. Поддерживаются три вида плагинов: ввода, фильтрации и вывода. В данный момент есть более 200 поддерживаемых плагинов и их количество постоянно увеличивается.

Logstash — отличный механизм передачи данных, который позволяет создавать масштабируемые контейнеры данных в реальном времени.

Beats

Beats — это легковесная платформа доставки данных с открытым исходным кодом, дополняющая Logstash. В отличие от Logstash, работающей на серверной стороне, Beats располагается на стороне клиента. В ее основе лежит библиотека libbeat, которая предоставляет API для передачи данных от источника, настройки ввода и реализации сбора данных. Beats устанавливается на устройства, которые не являются частью серверных компонентов, таких как Elasticsearch, Logstash или Kibana. Они размещаются на некластерных узлах, которые также иногда называют *пограничными узлами*.

Команда Elastic и сообщество разработчиков открытого ПО создали много компонентов Beats, например Packetbeat, Filebeat, Metricbeat, Winlogbeat, Auditbeat и Heartbeat. Filebeat используется для доставки лог-файлов с ваших серверов на централизованный сервер Logstash или Elasticsearch. Metricbeat предназначен для мониторинга серверов и периодического сбора показателей операционных систем и сервисов, запущенных на ваших серверах. На данный момент существует более 40 сообществ Beat, созданных для мониторинга Elasticsearch, Cassandra, веб-сервера Apache, производительности JVM и др. Вы также можете создать свой компонент на основе библиотеки libbeat, если не нашли подходящий.

Logstash и Beats подробно рассматриваются в главах 5 и 6.

Kibana

Kibana — инструмент визуализации для Elastic Stack, который поможет вам наглядно представить данные в Elasticsearch. Его также часто называют окном в Elastic Stack. В Kibana

предлагается множество вариантов визуализаций, таких как гистограмма, карта, линейные графики, временные ряды и др. Вы можете создавать визуализации буквально парой щелчков кнопкой мыши и исследовать свои данные в интерактивном виде. Кроме того, есть возможность создавать красивые панели управления, состоящие из различных визуализаций, делиться ими, а также получать высококачественные отчеты.

В Kibana также предусмотрены инструменты для управления и разработки. Вы можете управлять настройками X-Pack для обеспечения безопасности в Elastic Stack, а с помощью инструментов разработчика создавать и тестировать запросы REST API.

Мы рассмотрим все возможности Kibana в главе 7.

X-Pack

X-Pack добавляет важные функции, чтобы Elastic Stack был готов для запуска в работу. Среди них вы найдете настройки безопасности, мониторинга, уведомлений, отчетов и графических возможностей Elastic Stack.

Безопасность

Плагин безопасности X-Pack добавляет возможности аутентификации и авторизации в Elasticsearch и Kibana, таким образом доступ к данным будет только у определенных людей, и они смогут увидеть лишь то, что позволяют их настройки доступа. Плагин безопасности работает со всеми компонентами незаметно для пользователя.

В лицензированной версии плагин безопасности позволяет также настраивать уровни доступа для полей и документов.

Мониторинг

Используя X-Pack, вы можете настроить мониторинг кластеров Elasticsearch и Kibana. Доступны для выбора следующие варианты мониторинга: для кластеров, узлов и показателей на уровне индексов. Плагин мониторинга сохраняет историю производительности, чтобы вы могли сравнить текущие показатели с их значениями в различные периоды в прошлом. Доступны также функции планирования.

Отчеты

Плагин отчетности X-Pack позволяет создавать высококачественные отчеты, доступные для печати, на базе визуализаций Kibana. Вы можете сформировать расписание для создания отчетов или запускать их в зависимости от происходящих событий.

Уведомления

X-Pack характеризуется широкими возможностями добавления уведомлений: их можно выводить различными способами и настраивать для них различные условия. Система достаточно гибкая в отношении того, когда, как и кого уведомлять.

Вероятно, вам будет полезно получить информацию о нарушениях безопасности, например узнать о пяти неудачных попытках залогиниться в течение часа из разных мест. Или вы хотите знать, пользуется ли ваш товар популярностью в социальных сетях. Вы можете использовать всю мощь запросов Elasticsearch для выбора условий создания уведомлений.

В то же время у вас есть широкий выбор способов доставки уведомлений: по электронной почте, Slack, Hipchat, PagerDuty и др.

Построение графиков

Графики позволяют исследовать взаимосвязи ваших данных. В целом данные в Elasticsearch воспринимаются как двумерный (плоский) список записей без каких-либо связей. Возможность рассмотрения взаимосвязей открывает новые способы применения данных. С помощью графиков вы увидите записи с общими свойствами, например с выборкой по людям, местам, товарам или предпочтениям.

Графики можно строить благодаря API Graph и пользовательскому интерфейсу Kibana. В работе применяются распределенные запросы, масштабируемое индексирование и актуальные возможности Elasticsearch.

Мы подробнее рассмотрим X-Pack в главе 8.

Elastic Cloud

Elastic Cloud — это облачный сервис по управлению компонентами Elastic Stack, предоставляемый компанией Elastic (<https://www.elastic.co/>) — автором и разработчиком Elasticsearch и других компонентов Elastic Stack. Все компоненты продукта (помимо X-Pack и Elastic Cloud) созданы на базе открытого исходного кода. Компания Elastic обслуживает все компоненты Elastic Stack, проводит тренинги, выполняет разработку и предоставляет облачные сервисы.

Помимо Elastic Cloud, есть и другие облачные решения, доступные для Elasticsearch, например *Amazon Web Services (AWS)*. Основное преимущество Elastic Cloud в том, что он создан и обслуживается авторами Elasticsearch и других компонентов Elastic Stack.

Способы применения Elastic Stack

Компоненты Elastic Stack можно использовать для различных задач, список которых регулярно пополняется за счет появления новых плагинов и дополнений к существующим компонентам. Ниже приведены примеры самых распространенных вариантов применения, но, разумеется, ваш выбор не ограничивается только ими:

- анализ и безопасность логов;
- поиск по продуктам;
- анализ показателей;
- веб-поиск и поиск по сайту.

Рассмотрим детально каждый вариант.

Анализ и безопасность логов

Трио Elasticsearch, Logstash и Kibana в прошлом было хорошо известно под названием ELK-стек. Благодаря ему система Elastic Stack стала идеальной платформой для сбора и анализа логов в централизованном виде.

Перед командами поддержки приложений зачастую стоит непростая задача по управлению большим количеством приложений, размещенных на десятках или сотнях серверов. В инфраструктуру, созданную для работы этих приложений, могут входить следующие компоненты:

- веб-серверы;
- серверы приложений;
- серверы баз данных;

- брокеры сообщений.

Нередко приложения для бизнеса используют все эти типы серверов, причем в нескольких модификациях. При возникновении ошибки или другой проблемы команде поддержки нужно логиниться на каждом сервере для поиска неполадок. Это занимает немало времени, учитывая, что к тому же приходится просматривать лог-файлы в исходном, необработанном виде. Elastic Stack предоставляет полный набор инструментов для сбора, централизации, анализа логов, визуализации данных, добавления уведомлений и выведения отчетности по происходящим ошибкам. Каждый компонент играет свою роль в решении проблемы.

- Фреймворк Beats, а именно компонент Filebeat, может работать как упрощенный агент по сбору и переадресации логов.
- Logstash может централизовать все события, полученные из Beats, и обработать каждую запись логов, прежде чем отправлять в кластер Elasticsearch.
- Elasticsearch индексирует файлы регистрации. Теперь для обработанных файлов доступны и поиск, и аналитика.
- С помощью Kibana можно настроить визуализацию ошибок, предупреждений и другой информации из логов. Вы также можете создать панели управления для централизованного мониторинга происходящих событий в реальном времени.
- С X-Pack можно настроить безопасность решения и выдачу уведомлений. Этот инструмент позволяет создавать отчеты и анализировать связи между данными.

Как можно увидеть на примере, с помощью Elastic Stack реально построить полное решение для сбора и мониторинга логов.

Поиск по продуктам

Поиск по продуктам (товарам) заключается в выборе наиболее релевантных продуктов из тысяч или десятков тысяч вариантов и их выводе в списке в числе первых. Эта задача особенно актуальна для сайтов электронной коммерции, где продается огромное количество продуктов от множества различных производителей.

Полнотекстовый поиск в Elasticsearch и другие виды релевантного поиска помогают находить лучшие варианты. Отображение наиболее соответствующих условию продуктов на первой странице имеет особую ценность, поскольку повышает шансы на успешную продажу товара. Представим, что покупатель ищет аксессуары для iPhone 7, а в результатах поиска на первой странице видит различные чехлы, зарядные устройства и аксессуары для предыдущих поколений iPhone. Возможности анализа текста, предоставляемые Lucene, и нововведения Elasticsearch гарантируют получение требуемого результата поиска, в данном случае — аксессуаров для седьмой версии iPhone.

Однако задача релевантного поиска стоит не только перед сайтами электронной коммерции. Elasticsearch можно использовать для любого приложения, которому необходимо найти самый релевантный продукт из миллионов или миллиардов товаров.

Анализ показателей

У Elastic Stack есть богатые возможности аналитики благодаря многофункциональным API сбора информации в Elasticsearch. Таким образом, эта система является отличным инструментом для анализа данных и показателей. Показатели обычно представляют собой числовые значения, что отличает их от неструктурированного текста, характерного для документов и веб-страниц. Например, это могут быть данные с различных сенсоров, устройств Интернета вещей (IoT), показатели мобильных устройств, серверов, виртуальных машин, сетевых маршрутизаторов, сетевых коммутаторов и т.д.

Метрические данные, как правило, имеют временную природу, то есть значения или измерения записываются за определенный промежуток времени. Фиксируемые показатели чаще всего поступают с конкретного объекта. Например, сведения о температуре (метрические) записываются определенным устройством со своим идентификатором. Тип, название объекта, отдел, этаж и пр. — это данные, ассоциированные с этим показателем. Может также предоставляться информация о местоположении датчика, то есть широта и долгота.

Elasticsearch и Kibana позволяют разбивать метрические данные на фрагменты и распределять их в зависимости от разных параметров для более удобной работы. Elasticsearch имеет широкий функционал по управлению временными и пространственными данными — вы можете отобразить показатели, собранные из миллионов источников, в виде линейных графиков или гистограмм. Можно также выполнить пространственный анализ карты.

Создание приложения для анализа метрических данных мы рассмотрим в главе 9.

Веб-поиск и поиск по сайту

Elasticsearch может служить поисковой системой вашего сайта и выполнять поиск как в Google по всему его контенту. Поиск на таких платформах, как GitHub, Wikipedia, и многих других работает именно благодаря Elasticsearch.

Кроме того, Elasticsearch способна работать как платформа сбора контента. *Что такое контент-агрегатор, или платформа сбора контента?* Это инструмент для «прочесывания» сайтов, индексирования веб-страниц, предоставляющий поисковый функционал для найденного контента. Это мощное решение для построения агрегированных платформ.

Крупный веб-агрегатор Apache Nutch был создан Дагом Каттингом, одним из первых разработчиков Apache Lucene. Apache Nutch «прочесывает» сайты, обрабатывает HTML-страницы, сохраняет и индексирует их для поиска. Он дополняет возможности индексирования в Elasticsearch и Apache Solr для их поисковых систем.

Как вы можете видеть, Elasticsearch и Elastic Stack можно использовать для широкого спектра задач. Elastic Stack — это платформа с расширенным набором инструментов для создания комплексных решений поиска и аналитики. Она подходит для разработчиков, архитекторов, бизнес-аналитиков и системных администраторов. Вполне возможно создать решение на базе Elastic Stack, почти не прибегая к написанию кода, исключительно за счет изменения конфигурации. В то же время система Elasticsearch очень гибкая, следовательно, разработчики и программисты могут строить мощные приложения благодаря обширной поддержке языков программирования и REST API.

Скачивание и установка

Теперь, когда приведено достаточно причин для изучения Elasticsearch и Elastic Stack, начнем со скачивания и установки основных компонентов. Для начала скачаем и установим Elasticsearch и Kibana. Остальные компоненты мы установим тогда, когда они нам понадобятся в процессе изучения. Kibana нам нужен не только из-за визуализаций, но и потому, что в нем имеется пользовательский интерфейс для взаимодействия с Elasticsearch.

Начиная с версии Elastic Stack 5.x, все компоненты обновляются вместе и имеют один номер версии. К этому моменту они уже протестированы на совместимость друг с другом. Это касается и компонентов Elastic Stack версии 6.x.

На момент написания этой книги актуальная версия Elastic Stack — 6.0.0. Во всех примерах мы использовали именно эту версию и ее компоненты.

Установка Elasticsearch

Elasticsearch может быть скачана в виде файла ZIP, TAR, DEB или RPM. Если вы работаете в Ubuntu, Red Hat или CentOS Linux, возможна прямая установка через пакетные менеджеры apt или yum.

Мы будем использовать формат ZIP, поскольку это проще для целей разработки.

1. Перейдите по ссылке <https://www.elastic.co/downloads/elasticsearch> и скачайте ZIP-дистрибутив. Вы можете выбрать и более раннюю версию, если ищете конкретное издание.
2. Распакуйте файлы и перейдите в созданную папку. Запустите файлы bin/elasticsearch или bin/elasticsearch.bat.

3. Выполните curl http://localhost:9200 или откройте указанную ссылку в своем любимом браузере.

Вы должны увидеть код подобного вида (рис. 1.2).

```
$ curl http://localhost:9200?pretty
{
  "name" : "bhbP6GV",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "QQhvZ7YGTj0Rn5HcSsQf_Q",
  "version" : {
    "number" : "6.0.0",
    "build_hash" : "8f0685b",
    "build_date" : "2017-11-10T18:41:22.859Z",
    "build_snapshot" : false,
    "lucene_version" : "7.0.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
$
```

Рис. 1.2

Поздравляем! Вы только что установили единичный узел кластера Elasticsearch.

Установка Kibana

Утилита Kibana также доступна во множестве форматов: ZIP, TAR.GZ, RMP или DEB для устройств с 32- или 64-битной архитектурой.

1. Перейдите по ссылке <https://www.elastic.co/downloads/kibana> и скачайте архив ZIP или TAR.GZ для вашей платформы.

2. Распакуйте файлы и перейдите в созданную папку.
Запустите файл `bin/kibana` или `bin/kibana.bat`.
3. Откройте ссылку `http://localhost:5601` в своем любимом браузере.

Поздравляем! Теперь у вас есть рабочая установка Elasticsearch и Kibana.

Резюме

В этой главе мы поговорили о необходимости альтернативных технологий для поиска и анализа данных, отличных от реляционных баз данных и хранилищ NoSQL. Мы рассмотрели сильные стороны Elasticsearch, являющейся ядром Elastic Stack. Кратко ознакомились с остальными компонентами Elastic Stack и разобрались, какое место они занимают во всей экосистеме. Помимо этого, мы ознакомились с примерами практического использования Elastic Stack. Мы успешно скачали и установили Elasticsearch и Kibana для начала нашего путешествия в мир Elastic Stack.

В следующей главе мы начнем с изучения основных концепций Elasticsearch: разберемся в индексах, шардах, типах данных, разметке и других фундаментальных понятиях. Мы также поработаем с *CRUD-операциями* (*создание, чтение, обновление и удаление*) в Elasticsearch и попытаемся разобраться, что же такое поиск.

2. Начало работы с Elasticsearch

В первой главе мы поговорили о том, зачем изучать Elastic Stack, и рассмотрели примеры его использования.

В этой главе мы начнем наше путешествие по миру Elastic Stack с его самого главного компонента — Elasticsearch. Это движок для поиска и аналитики внутри Elastic Stack. Мы разберем основные понятия Elasticsearch на практических примерах, по пути узнав все необходимое о запросах, фильтрации и поиске.

В данной главе мы рассмотрим следующие темы.

- Работа с пользовательским интерфейсом (UI) Kibana Console.
- Основные понятия Elasticsearch.
- CRUD-операции.
- Создание индексов и контролирование разметки.
- Обзор REST API.

Пользовательский интерфейс Kibana Console

Прежде чем вы начнете писать первые запросы для работы с Elasticsearch, вам следует ознакомиться с важным инструментом: Kibana Console. Он пригодится при работе с REST API для любых возможных операций Elasticsearch. Kibana Console представляет собой удобный редактор, который поддерживает функцию автозавершения и форматирования запросов во время их написания.



Что такое REST API? REST означает Representational State Transfer. Это архитектурный стиль для взаимодействия систем друг с другом. REST развивался вместе с протоколом HTTP, и почти все системы, основанные на REST, используют HTTP как свой протокол. HTTP поддерживает различные методы: GET, POST, PUT, DELETE, HEAD и др. Например, GET предназначен для получения или поиска чего-либо, POST используется для создания нового ресурса, PUT может применяться для создания или обновления существующего ресурса, а DELETE – для безвозвратного удаления.

В главе 1 мы успешно установили Kibana и запустили пользовательский интерфейс по ссылке <http://localhost:5601>. Как уже говорилось, Kibana – это окно в Elastic Stack. Утилита не только позволяет визуализировать данные, но и содержит инструменты для разработчиков, такие как **Console** (Консоль). Пользовательский интерфейс консоли представлен на рис. 2.1.

The screenshot shows the Kibana interface with the title 'Console - Kibana'. The URL in the address bar is 'localhost:5601/app/kibana#/dev_tools/console?_g=()'. On the left, there's a sidebar with icons for Discover, Visualize, Dashboard, Timelion, Dev Tools (which is selected), and Management. The main area is titled 'Dev Tools' and 'Console'. In the 'Console' tab, there's a code editor window with the following content:

```
1 - [
2   "name": "bhbP6GV",
3   "cluster_name": "elasticsearch",
4   "cluster_uuid": "QQhvZ7YGTj0Rn5HcSsQf_Q",
5   "version": {
6     "number": "6.0.0",
7     "build_hash": "8f0f685b",
8     "build_date": "2017-11-10T18:41:22.859Z",
9     "build_snapshot": false,
10    "lucene_version": "7.0.1",
11    "minimum_wire_compatibility_version": "5.6.0",
12    "minimum_index_compatibility_version": "5.0.0"
13  },
14  "tagline": "You Know, for Search"
15 }
```

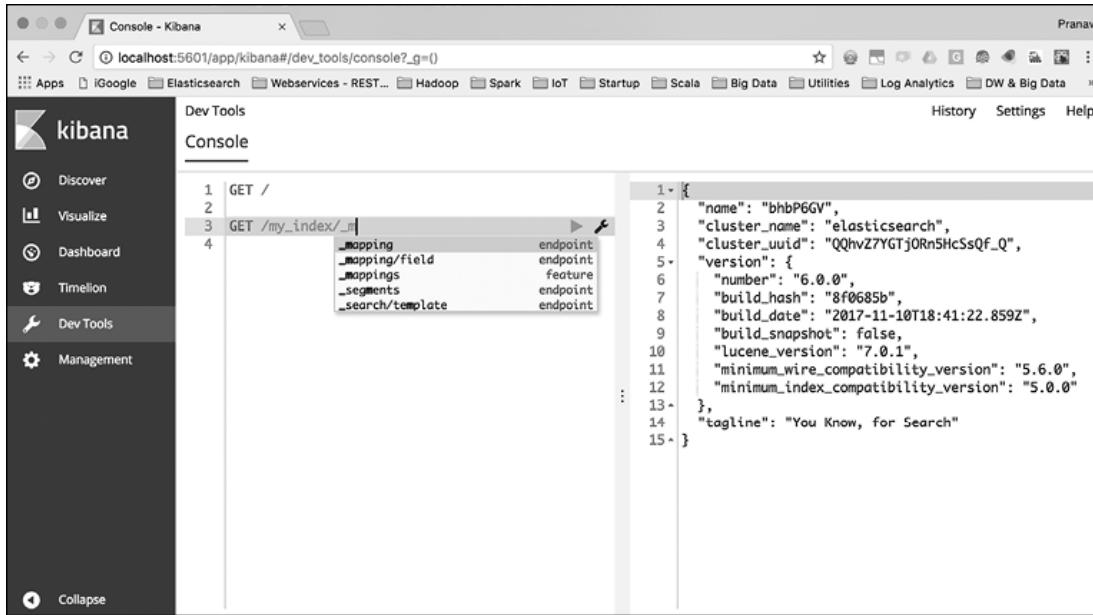
Рис. 2.1

После запуска Kibana нужно выбрать ссылку **Dev Tools** (Инструменты для разработчиков) на навигационной панели слева. Консоль разделена на две части: поле редактора и поле результатов. Можно вводить команды REST API, и после щелчка на зеленом треугольнике запрос будет отправлен в Elasticsearch (или кластер).

Здесь мы просто отправили запрос `GET /`. Это аналог команды `curl`, которую мы отправляли в Elasticsearch при проверке установки. Как видите, команда, отправленная через консоль, уже намного короче. Нам не нужно набирать `http`, хост и порт узла Elasticsearch, а именно `http://localhost:9200`. Но, как уже было сказано ранее, суть не только в том, что можно пропускать хост и порт при каждом запросе. Как только мы начинаем набирать текст в редакторе консоли, мы получаем список возможных команд (рис. 2.2).

Теперь, когда у нас есть правильный инструмент для генерирования и отправки запросов в Elasticsearch, перейдем к

изучению базовых понятий.



The screenshot shows the Kibana Dev Tools Console interface. On the left, there's a sidebar with icons for Discover, Visualize, Dashboard, Timelion, Dev Tools (which is selected), and Management. The main area has tabs for Dev Tools and Console. In the Dev Tools tab, a code editor shows a JSON object with various properties like name, cluster_name, and version. The code is as follows:

```
1 {  
2   "name": "bhbP6GV",  
3   "cluster_name": "elasticsearch",  
4   "cluster_uuid": "Q0QhvZ7YGTj0Rn5HcSsQf_Q",  
5   "version": {  
6     "number": "6.0.0",  
7     "build_hash": "8f0685b",  
8     "build_date": "2017-11-10T18:41:22.859Z",  
9     "build_snapshot": false,  
10    "lucene_version": "7.0.1",  
11    "minimum_wire_compatibility_version": "5.6.0",  
12    "minimum_index_compatibility_version": "5.0.0"  
13  },  
14  "tagline": "You Know, for Search"  
15}
```

Рис. 2.2

Основные понятия Elasticsearch

Чтобы работать с реляционными базами данных, нужно разбираться в таких понятиях, как строки, столбцы, таблицы и схемы. Elasticsearch и другие хранилища, ориентированные на документы, работают по иному принципу. Система Elasticsearch имеет четкую ориентацию на документы. Лучше всего для нее подходят JSON-документы. Они организованы с помощью различных типов и индексов. Далее мы рассмотрим ключевые понятия Elasticsearch:

- индекс;
- тип;
- документ;

- кластер;
- узел;
- шарды и копии;
- разметку и типы данных;
- обратный индекс.

Начнем с такого примера:

```
PUT /catalog/product/1
{
  "sku": "SP000001",
  "title": "Elasticsearch for Hadoop",
  "description": "Elasticsearch for Hadoop",
  "author": "Vishal Shukla",
  "ISBN": "1785288997",
  "price": 26.99
}
```

Скопируем этот пример в редактор консоли Kibana UI и выполним его. Таким образом мы проиндексируем документ, представляющий собой продукт в каталоге продуктов системы. Все примеры, приведенные для консоли Kibana UI, можно легко конвертировать в команды curl и выполнить в командной строке. Ниже показана версия curl для предыдущей команды консоли Kibana UI:

```
curl -XPUT
http://localhost:9200/catalog/product/1 -d '{'
"sku": "SP000001",
```

```
"title": "Elasticsearch for Hadoop",  
"description": "Elasticsearch for  
Hadoop", "author": "Vishal Shukla", "ISBN":  
"1785288997", "price": 26.99}'
```

Мы используем этот пример для знакомства с индексами, типами и документами.

В предыдущем фрагменте кода первая строка содержит команды PUT /catalog/product/1 и далее документ JSON.

PUT — HTTP-метод, используемый для индексирования нового документа. В нашем примере catalog — это имя индекса, product — имя типа, согласно которому документ будет индексироваться, а 1 — идентификатор, присвоенный документу после индексирования.

Индекс

Индекс — это контейнер, который в Elasticsearch хранит документы одного *типа* и управляет ими. Индекс может содержать документы одного типа, как показано на рис. 2.3.

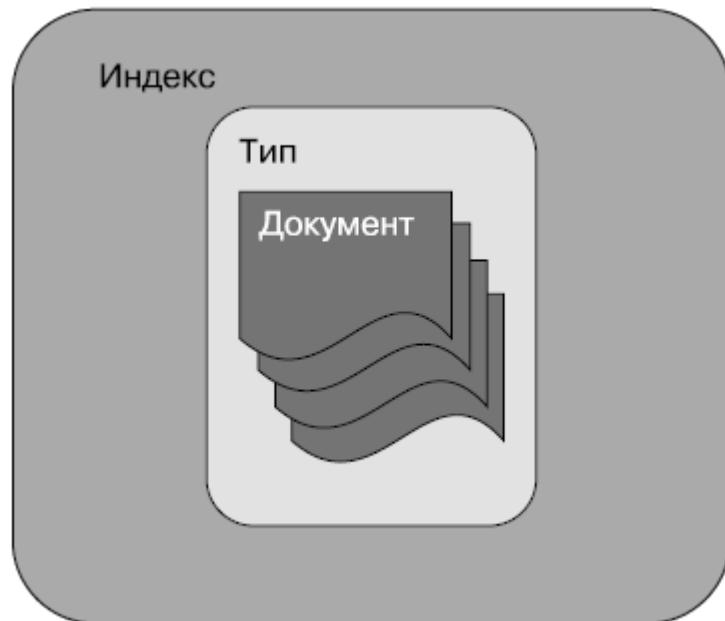


Рис. 2.3

Индекс — это логический контейнер типов. Одни параметры указываются на уровне индексов, другие — на уровне типов. Более наглядно эти различия мы рассмотрим в последующих разделах этой главы.

Индексы в Elasticsearch приблизительно аналогичны по структуре базе данных в реляционных базах данных. Продолжая аналогию, *тип* в Elasticsearch соответствует таблице, а *документ* — записи в ней. Но имейте в виду, что такую аналогию можно проводить сугубо для упрощения понимания. Отличие состоит в том, что в структуре реляционных баз данных почти всегда содержится несколько таблиц, а индекс может содержать только один тип.



До версии Elasticsearch 6.0 один индекс мог содержать несколько типов. Но начиная с версии 6.0 в пределах индекса допускается хранить только один тип. Если у вас уже есть индекс с несколькими типами, созданный в версии до 6.0, и вы обновляете его до версии 6.0, вы сможете и далее пользоваться своими старыми индексами. Но не получится создать индекс, содержащий более одного типа, в Elasticsearch 6.0 и выше.

Тип

В нашем примере с каталогом продуктов проиндексированный документ был типа «продукт». Каждый документ, сохраненный с этим типом, соответствует одному продукту. Один и тот же индекс не может иметь другие типы, такие как «клиенты», «заказы», «позиции заказа» и т.д. Типы помогают логически

группировать или организовывать однотипные документы по индексам.

Обычно документы с наиболее распространенным набором полей группируются под одним типом. Elasticsearch не требует наличия структуры, позволяя вам хранить любые документы JSON с любым набором полей под одним типом. На практике следует избегать смешивания разных сведений в одном типе, таких как «клиенты» и «продукты». Имеет смысл хранить их в разных типах и с разными индексами.

Документ

Как уже было сказано, JSON-документы лучше всего подходят для использования в Elasticsearch. Документ состоит из нескольких полей и является базовой единицей информации, хранимой в Elasticsearch. Например, у вас может быть документ, соответствующий одному продукту, одному клиенту или одной позиции заказа.

На рис. 2.3 показано, что документы хранятся внутри индексов и типов.

Документы содержат несколько *полей*. В документах JSON каждое поле имеет определенный *тип*. В примере с каталогом продуктов, который мы видели ранее, были поля `sku`, `title`, `description`, `price` и др. Каждое поле и его значение можно увидеть как пару «ключ — значение» в документе, где ключ — это имя поля, а значение — значение поля. Имя поля можно сопоставить с именем столбца в реляционных базах данных. Значение поля может восприниматься как значение столбца для выбранной строки, то есть значение выбранной ячейки в таблице.

В дополнение к пользовательским полям в документе Elasticsearch хранятся следующие внутренние метаполя:

- `_id` — уникальный идентификатор документа внутри типа по аналогии с первичным ключом в таблице базы данных. Он может генерироваться автоматически или выбираться пользователем;
- `_type` — это поле содержит тип документа;
- `_index` — хранит имя индекса документа.

Узел

Elasticsearch — распределенная система. Она состоит из множества процессов, запущенных на разных устройствах в сети и взаимодействующих с другими процессами. В главе 1 мы скачали, установили и запустили Elasticsearch. Таким образом мы запустили так называемый единичный узел кластера Elasticsearch.

Узел Elasticsearch — это единичный сервер системы, который может быть частью большого кластера узлов. Он участвует в индексировании, поиске и выполнении других операций, поддерживаемых Elasticsearch. Каждому узлу Elasticsearch в момент запуска присваиваются уникальный идентификатор и имя. Можно также назначить узлу статическое имя с помощью параметра `node.name` в конфигурационном файле Elasticsearch `config/elasticsearch.yml`.



Каждому узлу Elasticsearch соответствует основной конфигурационный файл, который находится в подкаталоге настроек. Формат файла YML (полное название — YAML Ain't Markup Language). Вы можете использовать этот файл для

изменения значений по умолчанию, таких как имя узла, порты, имя кластера.

На базовом уровне узел соответствует одному запущенному процессу Elasticsearch. Он отвечает за управление соответствующей ему частью данных.

Кластер

Кластер содержит один или несколько индексов и отвечает за выполнение таких операций, как поиск, индексирование и агрегации. Кластер формируется одним или несколькими узлами. Любой узел Elasticsearch всегда является частью кластера, даже если это кластер единичного узла. По умолчанию каждый узел пытается присоединиться к кластеру с именем Elasticsearch. Если вы запускаете несколько узлов внутри одной сети без изменения параметра `cluster.name` в файле `config/elasticsearch.yml`, они автоматически объединяются в кластер.



Рекомендуется изменять параметр `cluster.name` в конфигурационном файле Elasticsearch, чтобы избежать подключения к другому кластеру в той же сети. Согласно настройкам по умолчанию узел соединяется с существующим кластером внутри сети, и ваш локальный узел может попробовать соединиться с другим узлом для формирования кластера. Это может происходить на компьютерах разработчиков и другом оборудовании в том случае, если узлы находятся в одной сети.

Кластер состоит из нескольких узлов, каждый из которых отвечает за хранение своей части данных и управление ею. Один кластер может хранить один или несколько индексов. Индекс логически группирует разные типы документов.

Шарды и копии

Для начала разберемся, что такое кластер. Один индекс содержит документы одного или нескольких типов. Шарды помогают распределить индекс по кластеру. Они распределяют документы из одного индекса по различным узлам. Объем информации, который может храниться в одном узле, ограничивается дисковым пространством, оперативной памятью и вычислительными возможностями этого узла. Шарды помогают распределять данные одного индекса по всему кластеру и тем самым оптимизировать ресурсы кластера.

Процесс разделения данных по шардам называется *шардированием*. Это неотъемлемая часть Elasticsearch, необходимая для масштабируемой и параллельной работы с выполнением оптимизации:

- дискового пространства по разным узлам кластера;
- вычислительной мощности по разным узлам кластера.

По умолчанию каждый индекс настроен так, чтобы иметь пять шардов в Elasticsearch. В момент создания индекса можно обозначить количество шардов, на которые будут разделены данные вашего индекса. После того как индекс создан, количество шардов невозможно изменить.

На рис. 2.4 показано, как пять шардов могут быть распределены по кластеру из трех узлов.

На рисунке шарды обозначены символами от P1 до P5. Каждый шард содержит примерно 1/5 всех данных, хранимых в индексе. Когда происходит запрос по этому индексу, Elasticsearch проверяет все шарды и объединяет результат.

Теперь представим, что один из узлов (узел 1) вышел из строя. Вместе с ним мы теряем часть данных, которая хранилась в шардах P1 и P2 (рис. 2.5).

Распределенные системы наподобие Elasticsearch приспособлены к работе даже при неполадках оборудования. Для этого предусмотрены *реплики шардов*, или *копии*. Каждый шард индекса может быть настроен таким образом, чтобы у него было некоторое количество копий или не было ни одной. Реплики шардов — это дополнительные копии оригинального или первичного шарда для обеспечения высокого уровня доступности данных.

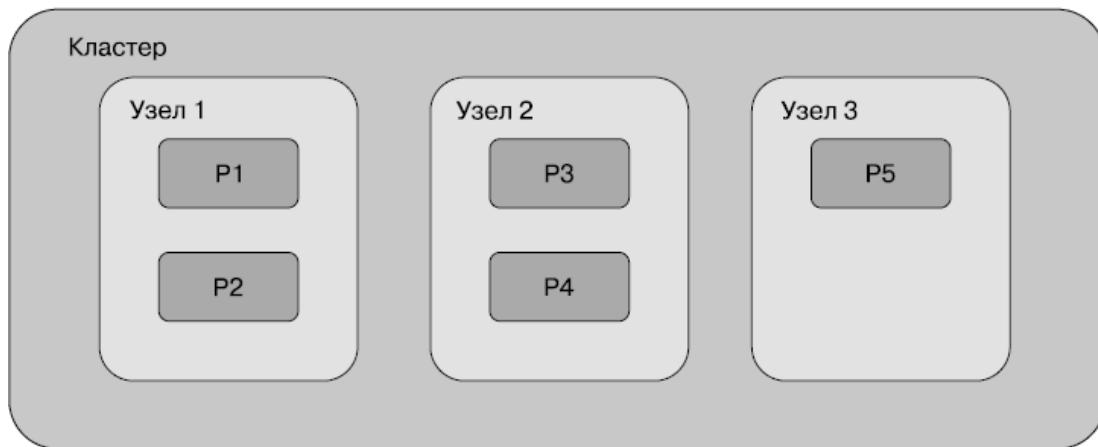


Рис. 2.4

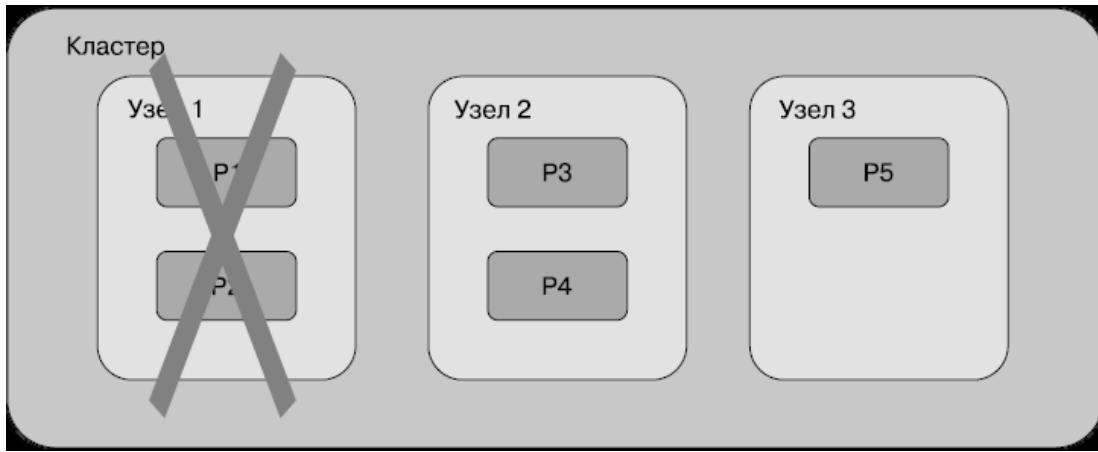


Рис. 2.5

На рис. 2.6 показаны пять первичных шардов и по одной копии на каждый шард.

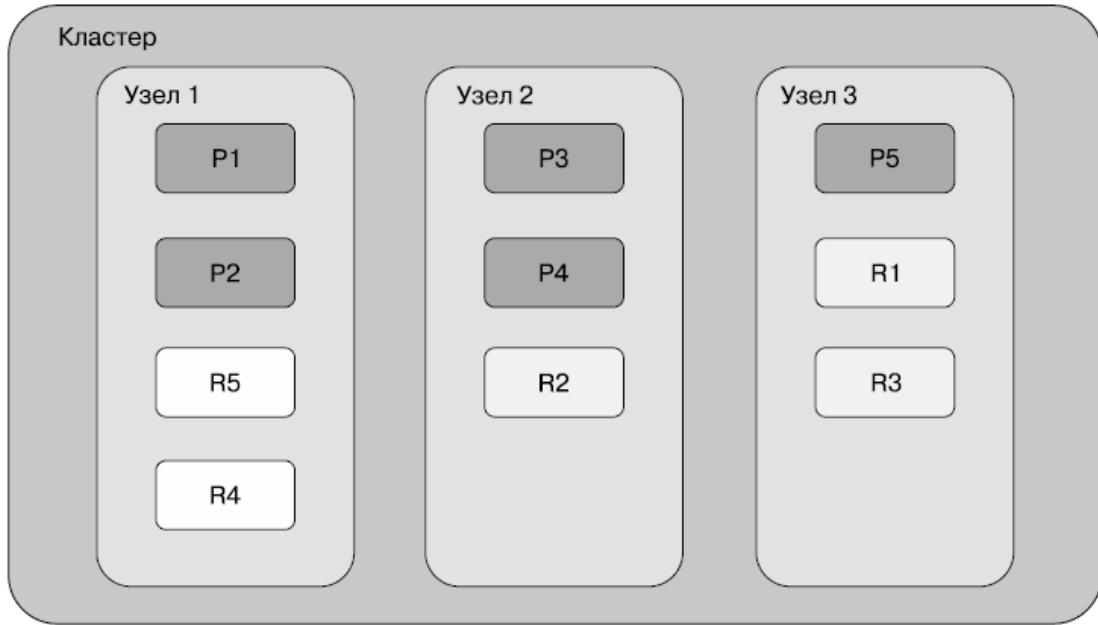


Рис. 2.6

Первичные шарды указаны темно-серым цветом, копии — светлым. При наличии копий, если узел 1 выходит из строя, все равно будут доступны все шарды в узлах 2 и 3. Реплики могут

становиться первичными шардами в тех случаях, когда соответствующие им шарды недоступны.

Копии шардов не только обеспечивают высокий уровень доступности данных, но и полезны для распределения запросов по копиям. Операции чтения данных, такие как поиск, запросы и агрегирование, могут выполняться и по копиям. Elasticsearch распределяет выполнение запросов по узлам кластера, в котором находятся копии.

Подводя итоги, можно сказать, что узлы соединяются для формирования кластера. Кластеры обеспечивают физический уровень для создания индексов. Индекс может содержать один или несколько типов, а каждый тип — миллионы или миллиарды документов. Индексы разделяются на шарды, которые являются фрагментами данных внутри индекса. Шарды распределяются по узлам кластера. Реплики — копии первичных шардов, обеспечивающие высокий уровень доступности данных при неполадках оборудования.

Разметка и типы данных

Elasticsearch — неструктурированная система, благодаря чему в ней можно хранить документы с любым количеством полей и типов полей. В реальности данные никогда не бывают абсолютно бесструктурными. Всегда есть некий набор полей, общий для всех документов этого типа. Фактически типы внутри индексов должны создаваться на основе общих полей. Обычно один тип документов внутри индекса содержит несколько общих полей.

Реляционные базы данных имеют четкую структуру. На момент создания таблицы необходимо разметить ее структуру именами столбцов и типами данных для каждого столбца.

Невозможно вставить запись с новым столбцом или другим типом данных во время работы с таблицей.

Важно понимать, какие типы данных поддерживает Elasticsearch.

Типы данных

Elasticsearch поддерживает широкий набор типов данных для различных сценариев хранения текстовых данных, чисел, булевых, бинарных объектов, массивов, объектов, вложенных типов, геоточек, геоформ и многих других специализированных типов данных, например адресов IPv4 и IPv6.

В документе каждое поле имеет ассоциированный тип данных. Ниже приведен полный список типов данных, поддерживаемых Elasticsearch.

Основные типы данных

Рассмотрим основные типы данных в Elasticsearch.

- **Строковые:**

- `text` — полезен для полнотекстового поиска по полям, которые содержат длинные текстовые значения. Эти поля анализируются до индексирования для поддержки полнотекстового поиска;
- `keyword` — позволяет выполнять анализ строковых полей. Поля такого типа поддерживают сортировку, фильтрацию и агрегацию.

- **Числовые:**

- `byte`, `short`, `integer` и `long` – целые числа со знаком с 8-, 16-, 32- и 64-битной точностью соответственно;
 - `float` и `double` – числа с плавающей точкой по стандарту IEEE 754 с 32-битной одиночной точностью и 64-битной двойной точностью;
 - `half-float` – число с плавающей точкой по стандарту IEEE 754 с 16-битной половинной точностью;
 - `scaled_float` – число с плавающей точкой (`float`), но хранится как `long` с помощью умножения на коэффициент масштабирования.
-
- **Даты:** `date`. Используется для представления полей даты. Если формат не указан, Elasticsearch пытается проанализировать поле даты, используя формат уууу-ММ-дд'T'НН:мм:ssз. Поддерживает хранение точных меток времени вплоть до миллисекунды.
 - **Булевые типы:** `boolean`. Известный тип данных для всех языков программирования. Используется для представления логических полей (`true` или `false`).
 - **Бинарный тип данных:** `binary`. Делает возможным хранение произвольных бинарных значений после выполнения кодировки Base64.
 - **Диапазон:** `integer_range`, `float_range`, `long_range`, `double_range` и `date_range`. Обозначает диапазоны целых чисел, плавающих чисел, длинных чисел и т.д.



scaled_float – очень полезный тип данных для хранения таких значений, как цены, которые всегда имеют ограниченное количество знаков после десятичной точки. Цена может указываться с коэффициентом масштабирования 100, следовательно, ценник \$10.98 может храниться как 1098 центов и обрабатываться как целое число. Тип scaled_float гораздо эффективнее для хранения, чем другие типы, поскольку целые числа лучше сжимаются.

Комплексные типы данных

В Elasticsearch поддерживаются следующие комплексные типы данных.

- **Массив** — определяет массивы одного типа. Например, массивы строковых данных, массивы целых чисел и т.д. В массивах не разрешается смешивать типы данных.
- **Объект** — допускает использование внутренних объектов в документах JSON.
- **Вложенный тип данных** — каждый объект в массиве индексируется как новый вложенный документ. Поскольку объекты обрабатываются внутри как отдельные документы, следует использовать специальный тип запроса для запроса вложенных документов.

Другие типы данных

Elasticsearch также поддерживает следующие типы данных.

- **Геоточка** — делает возможным хранение геоточек по широте и долготе. Например, тип данных «геоточка» полезен для таких запросов, как поиск по всем банкоматам в радиусе 2 км от указанной точки.
- **Геоформа** — используется для хранения геометрических форм, таких как полигоны, карты и пр. Тип данных «геоформа» позволяет выполнять запросы по поиску всех предметов определенной формы.
- **Тип данных IP** — используется для хранения адресов IPv4 и IPv6.

Разметка

Для понимания принципа работы разметки добавим в каталог еще один товар:

```
PUT /catalog/product/2
{
  "sku": "SP000002",
  "title": "Google Pixel Phone 32GB - 5 inch
display",
  "description": "Google Pixel Phone 32GB - 5
inch display (Factory
Unlocked US Version)",
  "price": 400.00,
  "resolution": "1440 x 2560 pixels",
  "os": "Android 7.1"
}
```

Скопируем данный пример в редактор консоли Kibana UI и выполним его.

Как видно, у товара есть много разных полей. Тем не менее некоторые поля являются общими для всех товаров.

Вспомните, что, в отличие от реляционных баз данных, в Elasticsearch нам не нужно назначать поля, которые будут частью каждого документа. Фактически нам даже не нужно создавать индекс с именем каталога. После того как первый документ типа «продукт» проиндексирован в каталоге индексов, Elasticsearch выполняет следующие действия:

- создает индекс с именем каталога;
- обозначает разметку для типа продукта.

Создание индекса с именем каталога

В первую очередь нужно создать индекс, поскольку он еще не существует. Индекс создается с количеством шардов, заданным по умолчанию. В Elasticsearch существует такое понятие, как *шаблон индексов*, который позволяет построить шаблон для любого индекса. Иногда индекс формируется на ходу, например, когда добавление первого документа автоматически приводит к созданию нового индекса. В таких случаях для него используется соответствующий шаблон. Таким образом, новые индексы создаются контролируемо, то есть с необходимым количеством шардов и разметкой типов.

Индексы можно создавать и заблаговременно. В Elasticsearch есть отдельный API для индексов (<https://www.elastic.co/guide/en/elasticsearch/reference/current/indices.html>), который работает с операциями уровня индекса. К ним относятся создание, удаление, получение индекса, создание разметки и др.

Установка разметки для типа продукта

Второй шаг включает в себя определение разметки (mappings) для типов продуктов. Это действие выполняется потому, что каталога типов не существует, пока не проиндексирован первый документ. Проведем аналогию с типами в реляционных базах данных. До того как добавить столбец, нужно создать таблицу. Когда создается таблица в РСУБД (реляционная система управления базами данных), мы обозначаем поля (столбцы) и их типы в выражении CREATE TABLE.

Когда первый документ индексируется в пределах типа, который еще не существует, Elasticsearch пытается обозначить типы данных для всех полей. Эта функция называется *динамической разметкой типов*. По умолчанию разметка типов в Elasticsearch включена.

Чтобы увидеть разметку типов товаров в индексе каталогов, выполните следующую команду в консоли Kibana UI:

```
GET /catalog/_mapping/product
```

Это пример Get Mapping API (<https://www.elastic.co/guide/en/elasticsearch/reference/current/indic-es-get-mapping.html>). Вы можете запросить разметку определенного типа, все типы в пределах индекса или нескольких индексов.

Вывод должен выглядеть следующим образом:

```
{
  "catalog": {
    "mappings": {
      "product": {
        "properties": {
```

```
    "ISBN": {
        "type": "text"
    },
    "author": {
        "type": "text"
    },
    "description": {
        "type": "text"
    },
    "price": {
        "type": "float"
    },
    "sku": {
        "type": "text"
    },
    "title": {
        "type": "text"
    }
}
```

На верхнем уровне JSON-ответа каталог представляет собой индекс, для которого была запрошена разметка. Дочерний продукт разметки означает, что эта разметка — для типа продуктов.

В результате мы получим немного другую разметку типов, чем показано в предыдущем коде. Как видите, тип данных `float` задан только для цены, остальные поля размечены как

текстовые. В реальности каждое поле типа данных `text` размечается следующим образом:

```
"field_name": {  
    "type": "text",  
    "fields": {  
        "keyword": {  
            "type": "keyword",  
            "ignore_above": 256  
        }  
    }  
}
```

Как вы можете заметить, каждое поле, установленное строкой, получило тип данных `text`. Текстовый тип данных делает возможным полнотекстовый поиск по полю. То же поле хранится как мультиполе и как `keyword`. Это позволяет выполнять не только полнотекстовый поиск, но и анализ данных (сортировку, агрегацию и фильтрацию) по одному и тому же полю.

Обратный индекс

Обратный индекс — это основная структура данных в Elasticsearch и любой другой системе, которая поддерживает полнотекстовый поиск. Обратный индекс похож на глоссарий в конце книги. Он размечает термы (определения, термины), имеющиеся в документах.

Например, вы можете построить обратный индекс из следующих строк (табл. 2.1).

Таблица 2.1

Идентификатор документа	Документ
1	Завтра воскресенье
2	Воскресенье – последний день недели
3	Выбор за вами

По результатам индексирования трех документов Elasticsearch построит структуру данных, которая будет выглядеть так, как показано в табл. 2.2. Такая структура данных называется *обратным индексом*.

Таблица 2.2

Терм (определение)	Частота	Документы (постинги)
вами	1	3
воскресенье	2	1, 2
выбор	1	3
день	1	2
завтра	1	1
неделя	1	2
последний	1	2

Обратите внимание на следующее.

- Документы разбиваются на определения без учета пунктуации и прописных букв.
- Термы сортируются по алфавиту.
- Столбец «Частота» показывает, как часто данное определение встречается во всем наборе документов.
- Третий столбец показывает, в каких документах было найдено определение. Кроме того, там могут содержаться

точные места нахождения (офсеты внутри документа).

При поиске определений содержащие их документы находятся довольно быстро. Если пользователь ищет определение «воскресенье», поиск по столбцу определений будет быстрым благодаря тому, что все они отсортированы в индексе. Даже если есть миллион определений, их легко и быстро найти, когда они отсортированы.

Представим сценарий, когда пользователь ищет два слова, например «последнее воскресенье». Обратный индекс может быть настроен на поиск отдельно слов «последнее» и «воскресенье». Документ 2 содержит оба определения, следовательно, это более подходящее соответствие, чем документ 1, содержащий только одно определение. Подобным образом легко узнать частоту появления определения внутри индекса.

Обратные индексы — это структурные блоки, из которых состоит быстрый поиск. Конечно, Elasticsearch использует и другие нововведения, помимо обычного обратного индекса, что обеспечивает хорошие возможности для поиска и аналитики.

По умолчанию Elasticsearch строит обратный индекс для всех полей документа, указывая на документ, в котором присутствует конкретное поле.

Операции CRUD

В этом разделе мы поговорим о том, как выполнять операции CRUD — основные операции для любого хранилища данных. В системе Elasticsearch операции CRUD нацелены на *документы*. Для их выполнения она предоставляет отличный REST API.

Для понимания принципа работы операций CRUD мы рассмотрим следующие API. Все они относятся к API

документов:

- Index API;
- Get API;
- Update API;
- Delete API.

Index API

В терминологии Elasticsearch добавление (или создание) документа в тип внутри индекса называется *операцией индексирования*. Чаще всего это добавление документа в индекс путем обработки всех полей внутри документа и постройки обратного индекса.

Есть два способа индексирования документа:

- индексирование с предоставлением идентификатора;
- индексирование без предоставления идентификатора.

Индексирование документа с предоставлением идентификатора

Мы уже встречались с таким видом операции индексирования. Пользователь может предоставить идентификатор документа, используя метод PUT.

Формат такого запроса следующий: PUT /<index>/<type>/<id> – с документом JSON в теле запроса:

```
PUT /catalog/product/1
{
    "sku": "SP000001",
```

```
"title": "Elasticsearch for Hadoop",
"description": "Elasticsearch for Hadoop",
"author": "Vishal Shukla",
"ISBN": "1785288997",
"price": 26.99
}
```

Индексирование документа без предоставления идентификатора

Если нет желания контролировать генерирование идентификаторов для документов, можно использовать метод POST.

Формат такого запроса следующий: POST /<index>/<type> – с документом JSON в теле запроса:

```
POST /catalog/product
{
  "sku": "SP000003",
  "title": "Mastering Elasticsearch",
  "description": "Mastering Elasticsearch",
  "author": "Bharvi Dixit",
  "price": 54.99
}
```

В данном случае идентификатор будет сгенерирован Elasticsearch. Он находится в выделенной хеш-строке:

```
{
  "_index": "catalog",
  "_type": "product",
  "_id": "AVrASKqgaBGmnAMj1SBe",
  "_version": 1,
```

```
"result": "created",
"_shards": {
  "total": 2,
  "successful": 1,
  "failed": 0
},
"created": true
}
```



По соглашениям REST POST используется для создания нового ресурса, а PUT – для обновления существующего ресурса. В нашем случае использование PUT означает следующее: «Я знаю, какой идентификатор хочу присвоить, следовательно, я указываю его в процессе индексирования документа».

Get API

Get API полезен для получения документа в тех случаях, когда уже известен его идентификатор. По сути, это операция GET по основному ключу:

```
GET /catalog/product/AVrASKqgaBGmnAMj1SBe
```

Формат запроса следующий: GET /<index>/<type>/<id>. Вывод должен быть таким:

```
{
  "_index": "catalog",
  "_type": "product",
  "_id": "AVrASKqgaBGmnAMj1SBe",
```

```
"_version": 1,  
"found": true,  
"_source": {  
    "sku": "SP000003",  
    "title": "Mastering Elasticsearch",  
    "description": "Mastering Elasticsearch",  
    "author": "Bharvi Dixit",  
    "price": 54.99  
}  
}
```

Update API

Update API используется для обновления существующего идентификатора документа.

Формат такого запроса следующий: POST <index>/<type>/<id>/_update – с документом JSON в теле запроса:

```
POST /catalog/product/1/_update  
{  
    "doc": {  
        "price": "28.99"  
    }  
}
```

Свойства, указанные в элементе doc, объединились в существующий документ. Стоимость предыдущей версии документа с идентификатором 1 составляла 26,99. Эта операция просто обновляет цену и оставляет остальные поля документа без изменений. Данный тип обновления означает, что doc

используется как часть документа для слияния с существующим документом. Поддерживаются и другие типы обновления.

Вывод по запросу обновления выглядит следующим образом:

```
{  
    "_index": "catalog",  
    "_type": "product",  
    "_id": "1",  
    "_version": 2,  
    "result": "updated",  
    "_shards": {  
        "total": 2,  
        "successful": 1,  
        "failed": 0  
    }  
}
```

Elasticsearch хранит версию каждого документа. При каждом обновлении документа номер версии увеличивается.

Продемонстрированное выше частичное обновление возможно только в том случае, если документ существовал изначально. Если документа с заданным идентификатором не существует, Elasticsearch выведет ошибку и сообщение о том, что документ отсутствует. Разберемся, как выполнить операцию *upsert*, используя Update API. Термин *upsert* в широком понимании означает *update* (обновить) или *insert* (вставить), то есть обновить документ, если он существует, или вставить новый документ — если нет.

Параметр *doc_as_upsert* проверяет, существует ли документ с предоставленным идентификатором и объединяет предоставленный элемент *doc* с существующим документом.

Если документ с таким идентификатором не существует, будет вставлен новый документ с необходимым содержимым.

В следующем примере мы используем `doc_as_upsert` для слияния с документом (идентификатор 3) или вставки нового документа, если он не существует:

```
POST /catalog/product/3/_update
{
  "doc": {
    "author": "Albert Paro",
    "title": "Elasticsearch 5.0 Cookbook",
    "description": "Elasticsearch 5.0 Cookbook Third Edition",
    "price": "54.99"
  },
  "doc_as_upsert
```

Мы можем обновить значение поля, базируясь на текущем значении этого или другого поля в документе. В следующем коде используется встроенный скрипт, увеличивающий стоимость определенного продукта на два:

```
POST
/catalog/product/AVrASKqgaBGmnAMj1SBe/_update
{
  "script": {
    "inline": "ctx._source.price += params.increment",
    "lang": "painless",
    "params": {
      "increment": 2
    }
  }
}
```

```
        }
    }
}
```

Поддержка скриптов позволяет прочитать текущее значение, увеличить его на какую-либо величину и сохранить в пределах одной операции. Этот встроенный скрипт создан с помощью собственного скриптового языка Elasticsearch. Синтаксическая конструкция для увеличения существующей переменной такая же, как в большинстве языков программирования.

Delete API

Delete API используется для удаления документа по идентификатору:

```
DELETE /catalog/product/AVrASKqgaBGmnAMj1SBe
```

Вывод операции удаления выглядит следующим образом:

```
{
  "found": true,
  "_index": "catalog",
  "_type": "product",
  "_id": "AVrASKqgaBGmnAMj1SBe",
  "_version": 4,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  }
}
```

Именно так Elasticsearch выполняет базовые операции CRUD. Имейте в виду, что Elasticsearch сохраняет данные в особой структуре, а именно в обратном индексе, используя возможности Apache Lucene. В реляционных базах данных используются B-дерева, которые лучше подходят для типовых CRUD-операций.

Создание индексов и контролирование разметки

В предыдущем разделе мы рассмотрели, как выполнять операции CRUD в Elasticsearch. В процессе мы также узнали, как индексирование первого документа по несуществующему индексу приводит к созданию нового индекса и разметки типа.

Чаще всего разработчики предпочитают не полагаться на автоматику, а самостоятельно контролировать, как создаются индексы и разметка. Сейчас мы узнаем, как взять контроль над этим процессом на себя, и рассмотрим следующие операции:

- создание индекса;
- создание разметки;
- обновление разметки.

Создание индекса

Вы можете создать индекс и указать количество шардов и копий:

```
PUT /catalog
{
  "settings": {
    "index": {
```

```
        "number_of_shards": 5,  
        "number_of_replicas": 2  
    }  
}  
}
```

Можно также указать разметку для типа во время создания индекса. Следующая команда создаст индекс с названием **catalog**, с пятью шардами и двумя копиями. Дополнительно мы также укажем тип с названием **my_type** с двумя полями, один типа **text** и второй — типа **keyword**:

```
PUT /catalog  
{  
    "settings": {  
        "index": {  
            "number_of_shards": 5,  
            "number_of_replicas": 2  
        }  
    },  
    "mappings": {  
        "my_type": {  
            "properties": {  
                "f1": {  
                    "type": "text"  
                },  
                "f2": {  
                    "type": "keyword"  
                }  
            }  
        }  
    }  
}
```

```
    }  
}  
}
```

Создание разметки типов в существующем индексе

Тип может быть добавлен внутри индекса после его создания.
Можно указать разметку типа следующим образом:

```
PUT /catalog/_mapping/category  
{  
  "properties": {  
    "name": {  
      "type": "text"  
    }  
  }  
}
```

Эта команда создает в существующем каталоге индекса тип под названием `category` с одним полем текстового типа.
Добавим несколько документов после создания нового типа:

```
POST /catalog/category  
{  
  "name": "books"  
}  
POST /catalog/category  
{  
  "name": "phones"  
}
```

После того как документы были индексированы, решено было добавить поля для хранения описания категории. Elasticsearch назначит тип автоматически, основываясь на том,

ЧТО ИМЕННО МЫ ВСТАВЛЯЕМ В НОВОЕ ПОЛЕ. Для предположения типа поля рассматривается лишь первое значение:

```
POST /catalog/category
{
  "name": "music",
  "description": "On-demand streaming music"
}
```

Когда новая категория проиндексирована с полями, назначается тип данных на основании его значения в исходном документе. Взглянем на разметку после индексирования документа:

```
{
  "catalog": {
    "mappings": {
      "category": {
        "properties": {
          "description": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          }
        },
        "name": {
          "type": "text"
        }
      }
    }
  }
}
```

```
        }
    }
}
}
```

Для описания поля был установлен тип данных `text`, задано поле под названием `keyword` одноименного типа. Это значит, что существует два поля: `description` и `description.keyword`. Поле `description` анализируется во время индексирования, в то время как `description.keyword` не анализируется и хранится без какой-либо оценки. По умолчанию поля индексируются в первый раз с двойными кавычками и хранятся одновременно как типы `text` и `keyword`.

Если нужно контролировать типы, следует указать разметку для поля до того, как проиндексируется первый документ, содержащий это поле. Тип поля не может быть изменен после того, как один или несколько документов были проиндексированы в пределах этого поля. Взглянем, как обновлять разметку для добавления поля необходимого типа.

Обновление разметки

Разметка для новых полей может быть добавлена после того, как создан тип. Разметка может быть обновлена с помощью API разметки PUT. Добавим поле `code` с типом `keyword` без анализа:

```
PUT /catalog/_mapping/category
{
  "properties": {
    "code": {
      "type": "keyword"
```

```
        }
    }
}
```

Эта разметка объединена в существующую разметку типа `category`. После слияния она выглядит следующим образом:

```
{
  "catalog": {
    "mappings": {
      "category": {
        "properties": {
          "code": {
            "type": "keyword"
          },
          "description": {
            "type": "text",
            "fields": {
              "keyword": {
                "type": "keyword",
                "ignore_above": 256
              }
            }
          },
          "name": {
            "type": "text"
          }
        }
      }
    }
  }
}
```

Любые последующие документы, которые проиндексированы с полем `code`, получают соответствующий тип данных:

```
POST /catalog/category
{
  "name": "sports",
  "code": "C004",
  "description": "Sports equipment"
}
```

Таким образом мы можем контролировать процесс создания индексов и назначения разметки типов, а также добавлять поля после создания типа.

Обзор REST API

Итак, мы разобрались, как выполнять базовые операции CRUD. Elasticsearch поддерживает широкий спектр типов операций. Одни операции выполняются с документами: создание, чтение, обновление, удаление и т.д. Другие обеспечивают поиск и агрегацию. Могут быть и операции уровня кластера, например мониторинг. В целом в Elasticsearch работают следующие виды API:

- API документов;
- API поиска;
- API агрегации;
- API индексов;
- API кластеров;

- CAT API.

В справочной документации Elasticsearch описаны все эти API, и в книге мы не будем углубляться в данную тему. Мы лишь рассмотрим несколько примеров их реального использования, а именно, как можно задействовать API для получения оптимальных результатов в Elasticsearch и других компонентах Elastic Stack.

В следующем разделе мы ознакомимся с общими правилами API, которые распространяются на все REST API.

Общие правила API

У всех REST API в Elasticsearch есть общие черты. Они работают схожим образом почти во всех API. Мы рассмотрим следующие функции:

- форматирование вывода JSON;
- управление несколькими индексами.

Форматирование вывода JSON

По умолчанию ответы на все запросы не форматируются. Вы получаете неотформатированную строку JSON:

```
curl -XGET  
http://localhost:9200/catalog/product/1
```

Вывод не отформатирован:

```
{"_index": "catalog", "_type": "product", "_id": "1"  
, "_version": 3,  
"found": true, "_source": {
```

```
"sku": "SP000001",
"title": "Elasticsearch for Hadoop",
"description": "Elasticsearch for Hadoop",
"author": "Vishal Shukla",
"ISBN": "1785288997",
"price": 26.99
}}
```

Для форматирования вывода нужно ввести pretty=true:

```
curl -XGET
http://localhost:9200/catalog/product/1?
pretty=true
{
  "_index" : "catalog",
  "_type" : "product",
  "_id" : "1",
  "_version" : 3,
  "found" : true,
  "_source" : {
    "sku" : "SP000001",
    "title" : "Elasticsearch for Hadoop",
    "description" : "Elasticsearch for Hadoop",
    "author" : "Vishal Shukla",
    "ISBN" : "1785288997",
    "price" : 26.99
  }
}
```

Надо отметить, что в консоли Kibana UI все выводы по умолчанию отформатированы.

Управление несколькими индексами

Можно выполнять такие операции, как поиск и агрегация, для нескольких индексов в одном запросе. Для поиска по конкретным индексам в запросе используются разные URL-ссылки. Разберемся, как можно применять URL для поиска в разных индексах и типах внутри них. Мы рассмотрим следующие сценарии работы с несколькими индексами внутри кластера:

- поиск по всем документам во всех индексах;
- поиск по всем документам в одном индексе;
- поиск по всем документам в одном типе одного индекса;
- поиск по всем документам в нескольких индексах;
- поиск по всем документам конкретного типа во всех индексах.

Следующий запрос соответствует поиску по всем документам. В данном случае реальное количество документов в выводе запроса будет ограничено десятью. По умолчанию ограничение `size` для результата равно 10, если в запросе не выбрано иное:

```
GET /_search
```

В выводе мы получим все документы из всех индексов данного кластера. Ответ будет похож на следующий, но здесь он сокращен, чтобы избежать бесполезного повторения документов:

```
{
```

```
"took": 3,  
"timed_out": false,  
"_shards": {  
    "total": 16,  
    "successful": 16,  
    </span>"failed": 0  
},  
"hits": {  
    "total": 4,  
    "max_score": 1,  
    "hits": [  
        {  
            "_index": ".kibana",  
            "_type": "doc",  
            "_id": "config:6.0.0",  
            "_score": 1,  
            "_source": {  
                "type": "config",  
                "config": {  
                    "buildNum": 16070  
                }  
            }  
        },  
        ...  
        ...  
    ]  
}
```

Очевидно, что это не самая полезная операция. Но мы рассмотрим ее для понимания результатов поиска:

- `took` — время, за которое ответил кластер, в миллисекундах;
- `timed_out: false` — строка означает, что операция прошла успешно, без превышения времени ожидания;
- `_shards` — показывает, сколько всего шардов на весь кластер было опрошено, успешно или нет;
- `hits` — количество соответствующих документов. Это *суммарное количество документов* по всем индексам, которое отвечает критериям поиска. Объект `max_score` показывает количество лучших соответствий поиску. Его дочерний объект `hits` выводит фактический перечень документов.



Список `hits` для массива данных не содержит все до единого соответствующие документы. Было бы неэффективно выводить абсолютно все, что отвечает критериям поиска, поскольку результаты могут исчисляться миллионами или миллиардами. Elasticsearch сокращает вывод согласно ограничению `size`, которое можно изменить по своему усмотрению, воспользовавшись параметром `GET /_search?size=100`. Значение по умолчанию равно 10, следовательно, исходный массив совпадений поиска будет содержать 10 записей.

Поиск по всем документам в одном индексе

С помощью следующей команды можно выполнять поиск по всем документам, но только в пределах индекса каталога:

```
GET /catalog/_search
```

Возможно также добавить фильтрацию по типу, а не только по индексу, например:

```
GET /catalog/product/_search
```

Поиск по всем документам в нескольких индексах

Выполнить поиск по всем документам в пределах индекса каталога и индекса под названием `my_index` можно с помощью такой команды:

```
GET /catalog,my_index/_search
```

Поиск по всем документам конкретного типа во всех индексах

Используя следующую команду, можно искать по всем индексам кластера, но только документы типа `product`:

```
GET/_all/product/_search
```

Эта функция может быть полезной, если у вас несколько индексов и каждый из них содержит *один и тот же тип*. Такой запрос поможет найти данные нужного типа из всех индексов.

Резюме

В этой главе мы познакомились с консолью Kibana UI, командой `curl` и узнали, как в Elasticsearch реализованы REST API. Мы также рассмотрели базовые понятия Elasticsearch. Мы

выполняли стандартные операции CRUD, необходимые для работы с любым хранилищем данных. Кроме того, детально рассмотрели такие операции, как создание индексов и разметки. Завершили главу обзором REST API в Elasticsearch, а также ознакомились с общими правилами, принятыми в большинстве API.

В следующей главе мы разберем, какие возможности поиска предоставляет Elasticsearch, и узнаем, как получить от поискового движка максимальную пользу.

3. Поиск — вот что важно

Одна из сильнейших сторон Elasticsearch — возможности поиска. В предыдущей главе мы подробно рассмотрели базовые концепции Elasticsearch, REST API и разобрались с основными операциями. С этими знаниями продолжим наш путь в освоении Elastic Stack. В данной главе будут рассмотрены следующие темы.

- Основы анализа текста.
- Поиск по структурированным данным.
- Написание сложных запросов.
- Полнотекстовый поиск.

Основы анализа текста

Анализ текста отличается от других видов анализа данных: чисел, даты и времени. Анализ числовых типов данных и даты/времени может выполняться определенным образом. Например, если вы ищете все записи, в которых указана цена 50 или выше, результат будет простым: «да» или «нет» для каждой записи. То есть либо запись соответствует запросу и выводу в результатах, либо нет. Аналогичным образом при поиске чего-либо по дате и времени критерий поиска четко обозначен — запись либо попадает в диапазон необходимых дат/времени, либо не попадает.

Однако анализ текстовых/строковых данных отличается. Текстовые данные могут иметь другую природу и могут

использоваться для структурированного и неструктурированного анализа.

Например, к структурированным типам для строковых полей относятся коды стран, коды продуктов, нечисловые серийные номера/идентификаторы и пр. Типы данных этих полей могут быть строковыми, но зачастую для них приходится писать запросы точного соответствия.

Сначала мы рассмотрим принципы анализа неструктурированного текста, который также известен как *полнотекстовый поиск*.

В предыдущей главе мы уже разобрались с такими понятиями Elasticsearch, как индексы, типы и разметка внутри типа. Все поля текстового типа анализируются инструментом под названием *анализатор*.

В следующих подразделах мы изучим такие темы.

- Анализаторы Elasticsearch.
- Использование встроенных анализаторов.
- Добавление автозавершения в пользовательском анализаторе.

Анализаторы Elasticsearch

Основная задача анализатора — взять значение поля и разбить его на термы, или определения (слова, словосочетания или фразы). В главе 2 мы рассмотрели структуру обратного индекса. Работа анализатора состоит в том, чтобы брать документы и каждое поле документа, а затем извлекать из них термы. Благодаря этим термам возможен поиск по индексу, таким образом, мы можем найти, какие именно документы содержат конкретные ключевые слова.

Анализатор проводит процесс разбивки введенных символов на термы. Это происходит дважды:

- во время индексирования;
- во время поиска.

Основная задача анализатора — обработать поля документа и построить актуальный индекс.

Каждое поле текстового типа должно быть проанализировано до того, как документ будет проиндексирован. Именно этот процесс анализа делает возможным поиск по документам с использованием любых поисковых терминов.

Анализаторы можно настроить для каждого поля, таким образом, мы можем иметь два поля текстового типа в пределах одного документа и каждое из них будет иметь отдельный анализатор.

Elasticsearch использует анализатор для обработки текстовых данных. В него входят следующие компоненты:

- *фильтры символов* — ноль или более;
- *токенизатор* — ровно один;
- *фильтры токенов* — ноль или более.

На следующей схеме показано, как эти компоненты расположены в анализаторе (рис. 3.1).

Рассмотрим назначение каждого компонента.

Фильтры символов

Во время настройки анализатора мы можем выбрать один или несколько фильтров символов либо не выбрать ни одного. Фильтр символов работает с потоком символов в поле ввода; каждый фильтр может добавлять, удалять или изменять символы в поле ввода.

Elasticsearch поставляется со встроенными фильтрами символов, которые можно сразу использовать. Либо можно создать свой собственный анализатор.

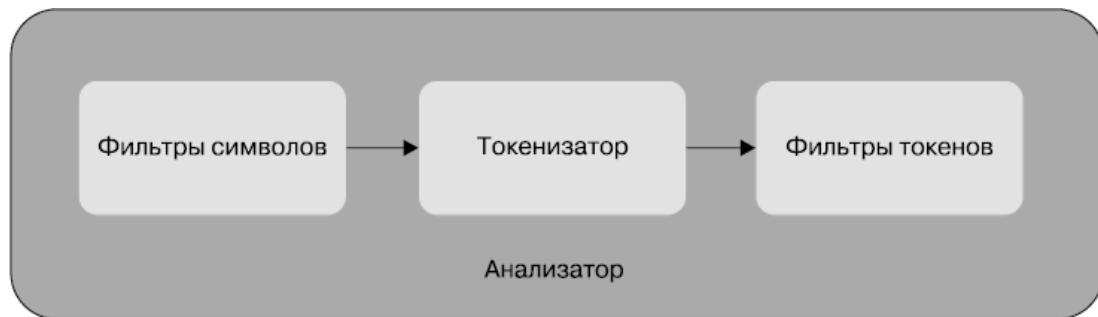


Рис. 3.1

Например, один из встроенных фильтров символов Elasticsearch называется *Mapping Char Filter*. Он может преобразовывать символ или последовательность символов в нужные вам символы.

Например, вы хотите превратить смайлики в текст, который описывал бы их смысл:

- :) должен быть преобразован в `_smile_` (улыбка);
- :(должен быть преобразован в `_sad_` (грусть);
- :D должен быть преобразован в `_laugh_` (смех).

Это можно сделать с помощью *Mapping Char Filter*:

```
"char_filter": {
```

```
"my_char_filter": {  
    "type": "mapping",  
    "mappings": [  
        ":)" => "_smile_",  
        ":(" => "_sad_",  
        ":D => _laugh_"  
    ]  
}  
}
```

Если данный фильтр символов использовать для создания анализатора, мы получим следующий результат:

Доброе утро всем :) будет преобразовано в Доброе утро всем _улыбка_.

Сегодня я себя плохо чувствую :(будет преобразовано в Сегодня я себя плохо чувствую _грусть_.

Поскольку фильтры символов находятся в самом начале цепи обработки, выполняемой анализатором (см. рис. 3.1), токенизатор всегда видит замененные символы. Фильтры символов удобны для замены символов чем-нибудь более понятным в определенных случаях, например замены числовых символов из других языков на понятные в вашей среде. Иначе говоря, числа из таких языков, как хинди, арабский и др., могут быть преобразованы в ноль, один, два и т.д.

Вы можете найти список доступных встроенных фильтров символов по следующей ссылке:
<https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-charfilters.html>.

Токенизатор

В анализаторе всегда есть один токенизатор. В его задачу входят получение потока символов и создание потока токенов. Эти токены используются для постройки обратного индекса. Токен примерно соответствует слову. В дополнение к разбику символов на слова или токены токенизатор также создает стартовые и конечные точки для каждого токена в потоке ввода.

Elasticsearch поставляется с несколькими токенизаторами, которые могут быть использованы для создания собственных анализаторов. Сама Elasticsearch также применяет эти токенизаторы для составления встроенных анализаторов.

Список доступных встроенных токенизаторов вы можете найти по ссылке <https://www.elastic.co/guide/en/elasticsearch/reference/current/analyzers-tokenizers.html>.

Стандартный токенизатор — один из самых популярных, поскольку подходит для большинства языков. Взглянем поближе на его возможности.

Стандартный токенизатор. В упрощенном понимании стандартный токенизатор разбивает поток символов, разделяя их знаками пунктуации и пробелами.

В следующем примере вы можете видеть, как стандартный токенизатор разбивает поток символов на токены:

```
POST _analyze
{
  "tokenizer": "standard",
  "text": "Tokenizer breaks characters into tokens!"
}
```

Данная команда приводит к следующему выводу. Обратите внимание на объекты `start_offset`, `end_offset` и `positions`:

```
{  
  "tokens": [  
    {  
      "token": "Tokenizer",  
      "start_offset": 0,  
      "end_offset": 9,  
      "type": "<ALPHANUM>",  
      "position": 0  
    },  
    {  
      "token": "breaks",  
      "start_offset": 10,  
      "end_offset": 16,  
      "type": "<ALPHANUM>",  
      "position": 1  
    },  
    {  
      "token": "characters",  
      "start_offset": 17,  
      "end_offset": 27,  
      "type": "<ALPHANUM>",  
      "position": 2  
    },  
    {  
      "token": "into",  
      "start_offset": 28,  
      "end_offset": 32,
```

```
        "type": "<ALPHANUM>" ,  
        "position": 3  
    },  
    {  
        "token": "tokens" ,  
        "start_offset": 33 ,  
        "end_offset": 39 ,  
        "type": "<ALPHANUM>" ,  
        "position": 4  
    }  
]  
}
```

Если необходимо, этот поток токенов может быть обработан фильтрами токенов анализатора.

Фильтры токенов

В анализаторе может быть один или несколько фильтров токенов либо не быть их вовсе. Каждый фильтр токенов может добавлять, удалять или изменять токены во входящем потоке токенов. Поскольку в анализаторе допускается наличие нескольких фильтров токенов, вывод каждого из них отправляется следующему, пока не будут пройдены все.

Elasticsearch поставляется с несколькими фильтрами токенов, и вы можете использовать их для создания собственного анализатора.

Вот несколько примеров встроенных фильтров токенов.

- *Фильтр токенов нижнего регистра.* Заменяет все токены на вводе версиями в нижнем регистре.

- *Стоп-токен-фильтр.* Убирает стоп-слова, то есть слова, которые не имеют особого значения для контекста. Например, в английском языке это слова *is*, *a*, *an* и *the*, которые не несут особого смысла в предложении. Для решения многих проблем текстового поиска имеет смысл убирать такие слова.

Список доступных встроенных фильтров токенов можно найти по следующей ссылке:
<https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-tokenfilters.html>.

Итак, мы рассмотрели назначение фильтров символов, токенизаторов и фильтров токенов. Следовательно, вы готовы узнать, как работают встроенные анализаторы Elasticsearch.

Использование встроенных анализаторов

Elasticsearch поставляется с несколькими встроенными анализаторами, которые вы можете использовать напрямую. Почти все из них сразу готовы к работе и не требуют изменений в конфигурации, но при этом они достаточно гибкие для настройки различных параметров.

Вот некоторые популярные анализаторы, поставляемые в комплекте с Elasticsearch.

- *Стандартный анализатор.* Анализатор по умолчанию. Если не настроены дополнительные анализаторы уровня поля, типа или индекса, он будет анализировать все поля.
- *Языковые анализаторы.* Разные языки ориентируются на разные правила грамматики. Следовательно, различаются и виды потока символов, а также преобразования их в токены

или слова. У каждого языка также есть свой настраиваемый список стоп-слов.

- *Анализатор пробелов.* Этот компонент разбивает ввод на токены в тех случаях, если в тексте найден пробел, отступ, перенос строки.

Вы можете найти список доступных встроенных фильтров токенов по следующей ссылке:
<https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis-analyzers.html>.

Стандартный анализатор. Подходит для многих языков и ситуаций. Кроме того, вы можете настроить его под конкретный язык или ситуацию. Он состоит из следующих компонентов.

- *Токенизатор: стандартный токенизатор.* Для разделения токенов по пробелам.
- *Фильтры токенов.*
 - *Стандартный фильтр токенов.* Применяется как метка-заполнитель в пределах стандартного анализатора. Не изменяет входящие токены, но может быть использован в будущем для выполнения задач.
 - *Фильтр токенов нижнего регистра.* Преобразует все токены ввода в нижний регистр.
 - *Стоп-токен-фильтр.* Убирает указанные стоп-слова. По умолчанию в качестве списка стоп-слов задано `_none_`, следовательно, никакие стоп-слова не удаляются.

Взглянем на работу стандартного анализатора с настройками по умолчанию:

```
PUT index_standard_analyzer
{
  "settings": {
    "analysis": {
      "analyzer": {
        "std": {
          "type": "standard"
        }
      }
    }
  },
  "mappings": {
    "my_type": {
      "properties": {
        "my_text": {
          "type": "text",
          "analyzer": "std"
        }
      }
    }
  }
}
```

Здесь создан индекс `index_standard_analyzer`. Обратите внимание на две вещи.

- Под элементом `settings` мы явно указали один анализатор с именем `std`. Тип анализатора – `standard`. Кроме этой,

никаких других настроек стандартного анализатора мы не выполняли.

- В индексе мы создали один тип с названием `my_type` и явно указали для анализатора единственное поле — `my_text`.

Посмотрим, как Elasticsearch станет выполнять анализ поля `my_text` каждый раз, когда любой документ в пределах этого индекса будет проиндексирован. Мы можем провести подобный тест с помощью API `_analyze`:

```
POST index_standard_analyzer/_analyze
{
  "field": "my_text",
  "text": "The Standard Analyzer works this
way."
}
```

Данная команда выведет следующие токены:

```
{
  "tokens": [
    {
      "token": "the",
      "start_offset": 0,
      "end_offset": 3,
      "type": "<ALPHANUM>",
      "position": 0
    },
    {
      "token": "standard",
      "start_offset": 4,
```

```
        "end_offset": 12,
        "type": "<ALPHANUM>",
        "position": 1
    },
{
    "token": "analyzer",
    "start_offset": 13,
    "end_offset": 21,
    "type": "<ALPHANUM>",
    "position": 2
},
{
    "token": "works",
    "start_offset": 22,
    "end_offset": 27,
    "type": "<ALPHANUM>",
    "position": 3
},
{
    "token": "this",
    "start_offset": 28,
    "end_offset": 32,
    "type": "<ALPHANUM>",
    "position": 4
},
{
    "token": "way",
    "start_offset": 33,
    "end_offset": 36,
    "type": "<ALPHANUM>",
    "position": 5
}
```

```
        }
    ]
}
```

Обратите внимание, что в данном случае в качестве анализатора уровня поля для `my_field` был явно выбран стандартный анализатор. Даже если не оставлять подобного указания, стандартный анализатор будет анализатором по умолчанию при условии, что не было указано иных вариантов.

Как видите, все токены в выводе записаны в нижнем регистре. Несмотря на то что стандартный анализатор имеет стоп-токен-фильтр, ни один из токенов не был отфильтрован. Поэтому в выводе `_analyze` все слова являются токенами.

Создадим другой индекс, который будет использовать стоп-слова английского языка:

```
PUT index_standard_analyzer_english_stopwords
{
  "settings": {
    "analysis": {
      "analyzer": {
        "std": {
          "type": "standard",
          "stopwords": "_english_"
        }
      }
    }
  },
  "mappings": {
    "my_type": {
      "properties": {
        "my_text": {

```

```
        "type": "text",
        "analyzer": "std"
    }
}
}
}
}
```

Обратите внимание на разницу. Новый индекс использует стоп-слова `_english_`. Вы можете указать список стоп-слов напрямую, например `stopwords: (a, an, the)`. Значение `_english_` включает все подобные слова на английском языке.

Когда вы попробуете использовать API `_analyze` в новом индексе, увидите, что он удаляет такие стоп-слова, как `the` и `this`:

```
POST
index_standard_analyzer_english_stopwords/_analyze
{
    "field": "my_text",
    "text": "The Standard Analyzer works this
way."
}
```

Вы получите ответ следующего вида:

```
{
    "tokens": [
        {
            "token": "standard",
            "start_offset": 4,
            "end_offset": 12,
```

```
        "type": "<ALPHANUM>",
        "position": 1
    },
    {
        "token": "analyzer",
        "start_offset": 13,
        "end_offset": 21,
        "type": "<ALPHANUM>",
        "position": 2
    },
    {
        "token": "works",
        "start_offset": 22,
        "end_offset": 27,
        "type": "<ALPHANUM>",
        "position": 3
    },
    {
        "token": "way",
        "start_offset": 33,
        "end_offset": 36,
        "type": "<ALPHANUM>",
        "position": 5
    }
]
}
```

Как видите, были удалены английские стоп-слова *the* и *this*. Таким образом, с небольшими настройками можно использовать стандартный анализатор для английского и многих других языков.

Рассмотрим практический пример создания собственного анализатора.

Добавление автозавершения в пользовательском анализаторе

В определенных ситуациях может понадобиться создать свой нестандартный анализатор путем выбора фильтров, токенизаторов и фильтров токенов. Создадим анализатор, который сможет предоставить такой функционал, как автозавершение текста.

Для добавления автозавершения мы не можем использовать стандартный анализатор или один из встроенных анализаторов Elasticsearch. Анализатор отвечает за создание термов во время индексирования. Наш анализатор должен уметь создавать такие термы, которые помогут с автозавершением. Для облегчения понимания перейдем к конкретному примеру.

Если бы во время индексирования мы использовали стандартный анализатор, то для поля со значением `Learning Elastic Stack 6` получили бы следующий вывод:

```
GET /_analyze
{
  "text": "Learning Elastic Stack 6",
  "analyzer": "standard"
}
```

Ответ на такой запрос будет содержать слова `Learning`, `Elastic`, `Stack` и `6`. Именно такие термы создала бы система Elasticsearch, сохранив их в индексе при использовании стандартного анализатора. Теперь нужно сделать следующее: когда пользователь начнет набирать отдельные буквы слова, программа должна предлагать подходящие результаты. Например, если пользователь набрал `elas`, он должен получить

такой результат: Learning Elastic Stack 6. Составим анализатор, который мог бы генерировать термы типа el, ela, elas, elast, elasti, elastic, le, lea и т.д.:

```
PUT /custom_analyzer_index
{
  "settings": {
    "index": {
      "analysis": {
        "analyzer": {
          "custom_analyzer": {
            "type": "custom",
            "tokenizer": "standard",
            "filter": [
              "lowercase",
              "custom_edge_ngram"
            ]
          }
        },
        "filter": {
          "custom_edge_ngram": {
            "type": "edge_ngram",
            "min_gram": 2,
            "max_gram": 10
          }
        }
      }
    }
  },
  "mappings": {
    "my_type": {
```

```
"properties": {
    "product": {
        "type": "text",
        "analyzer": "custom_analyzer",
        "search_analyzer": "standard"
    }
}
```

В данном случае в индексе работает нестандартный анализатор, который использует стандартный токенизатор для создания токенов, и два фильтра токенов — фильтр нижнего регистра и фильтр токенов `edge_ngram`. Последний разбивает каждый токен на два, три, четыре символа вплоть до десяти. При входящем токене-слове `elastic` будут созданы токены `el`, `ela` и т.д. Это позволит осуществлять поиск с автозавершением.

При условии, что оба продукта проиндексированы и пользователь уже ввел `Ela`, поиск должен вывести оба продукта:

```
POST /custom_analyzer_index/my_type
{
    "product": "Learning Elastic Stack 6"
}
```

```
POST /custom_analyzer_index/my_type
{
    "product": "Mastering Elasticsearch"
}
```

```
GET /custom_analyzer_index/_search
{
  "query": {
    "match": {
      "product": "Ela"
    }
  }
}
```

Поскольку индекс содержит термы el, ela и т.д., мы получим ответ в виде обоих названий. Это было бы невозможным при использовании стандартного анализатора в момент индексирования. На текущий момент можно считать, что стандартный анализатор (настроенный как `search_analyzer`) работает с учетом имеющихся критериев поиска, а полученный вывод использует для выполнения поиска. В этом примере он начнет искать термин ela в индексе. Поскольку индекс создан с использованием нестандартного анализатора с фильтром токенов `_ngram`, будут найдены соответствия для обоих названий.

Итак, анализаторы играют важную роль в функционировании Elasticsearch. Они решают, какие именно выражения сохраняются в индексе. Таким образом, от типа применяемого во время индексирования анализатора зависит, какие виды поисковых операций возможны с этим индексом. Например, стандартный анализатор не подойдет, если требуется поддержка функции автозавершения.

Прежде чем мы перейдем к следующему разделу и рассмотрим другие типы запросов, создадим необходимый для дальнейшей работы индекс и добавим нужные данные. Мы будем использовать данные каталога товаров популярного сервиса электронной коммерции www.amazon.com. Их можно

скачать по ссылке <http://dbs.uni-leipzig.de/file/Amazon-GoogleProducts.zip>.

Для начала создадим индекс и импортируем данные:

```
PUT /amazon_products
{
  "settings": {
    "number_of_shards": 1,
    "number_of_replicas": 0,
    "analysis": {
      "analyzer": {
        }
      }
    },
  "mappings": {
    "products": {
      "properties": {
        "id": {
          "type": "keyword"
        },
        "title": {
          "type": "text"
        },
        "description": {
          "type": "text"
        },
        "manufacturer": {
          "type": "text",
          "fields": {
            "raw": {
              "type": "keyword"
            }
          }
        }
      }
    }
  }
}
```

```
        }
      }
    },
    "price": {
      "type": "scaled_float",
      "scaling_factor": 100
    }
  }
}
```

Нужно проанализировать поля названия и описания — текстовые поля. Таким образом мы сделаем их доступными для полнотекстового поиска. Поле производителя является текстовым, но также присутствует поле с необработанным названием. Это поле сохраняется в двух видах: как `text` (`manufacturer`) и `keyword` (`manufacturer.raw`). Все поля типа `keyword` на внутреннем уровне используют анализатор ключевых слов, который состоит из кольцевого токенизатора ключевых слов. Он попросту возвращает весь ввод как один токен. Помните, что наличие фильтров символов и токенов в анализаторе не обязательно. Таким образом, указывая для поля тип `keyword`, мы выбираем кольцевой анализатор и тем самым пропускаем для этого поля весь процесс анализа.

Для поля цены выбран тип `scaled_float`. Это новый тип, появившийся в версии Elastic 6.0. Он на внутреннем уровне хранит переменные как целые числа с возможностью масштабирования. Например, 13.99 будет храниться как 1399 с фактором масштабирования 100. Это экономит занимаемую память, так как типы данных `float` и `double` занимают больше места.

Для импортирования данных проследуйте по инструкциям в сопроводительной информации и исходном коде к этой книге, которые размещены в репозитории GitHub по следующей ссылке: <https://github.com/pranav-shukla/learningelasticsearch>. Инструкции по импортированию находятся в файле chapter-03/README.md.

Выполнив нужные действия, убедитесь, что импортировали данные, с помощью следующего запроса:

```
GET /amazon_products/products/_search
{
  "query": {
    "match_all": {}
  }
}
```

В следующем разделе мы поговорим о запросах структурированного поиска.

Поиск по структурированным данным

В некоторых ситуациях требуется узнать, следует включать определенный документ в результаты поиска или нет, и для этого подойдет обычный бинарный ответ. С другой стороны, есть другие типы запросов, основанные на релевантности. Эти запросы возвращают оценку для каждого документа, чтобы обозначить, насколько хорошо он вписывается в запрос. Большинство структурированных запросов не нуждаются в оценке релевантности, и ответ всегда представляет собой простое «да/нет» для любого пункта, который включен в результаты или исключен из них. Эти структурированные

запросы также называют *запросами на уровне термов* (*term level queries*).

Рассмотрим порядок выполнения такого запроса (рис. 3.2).

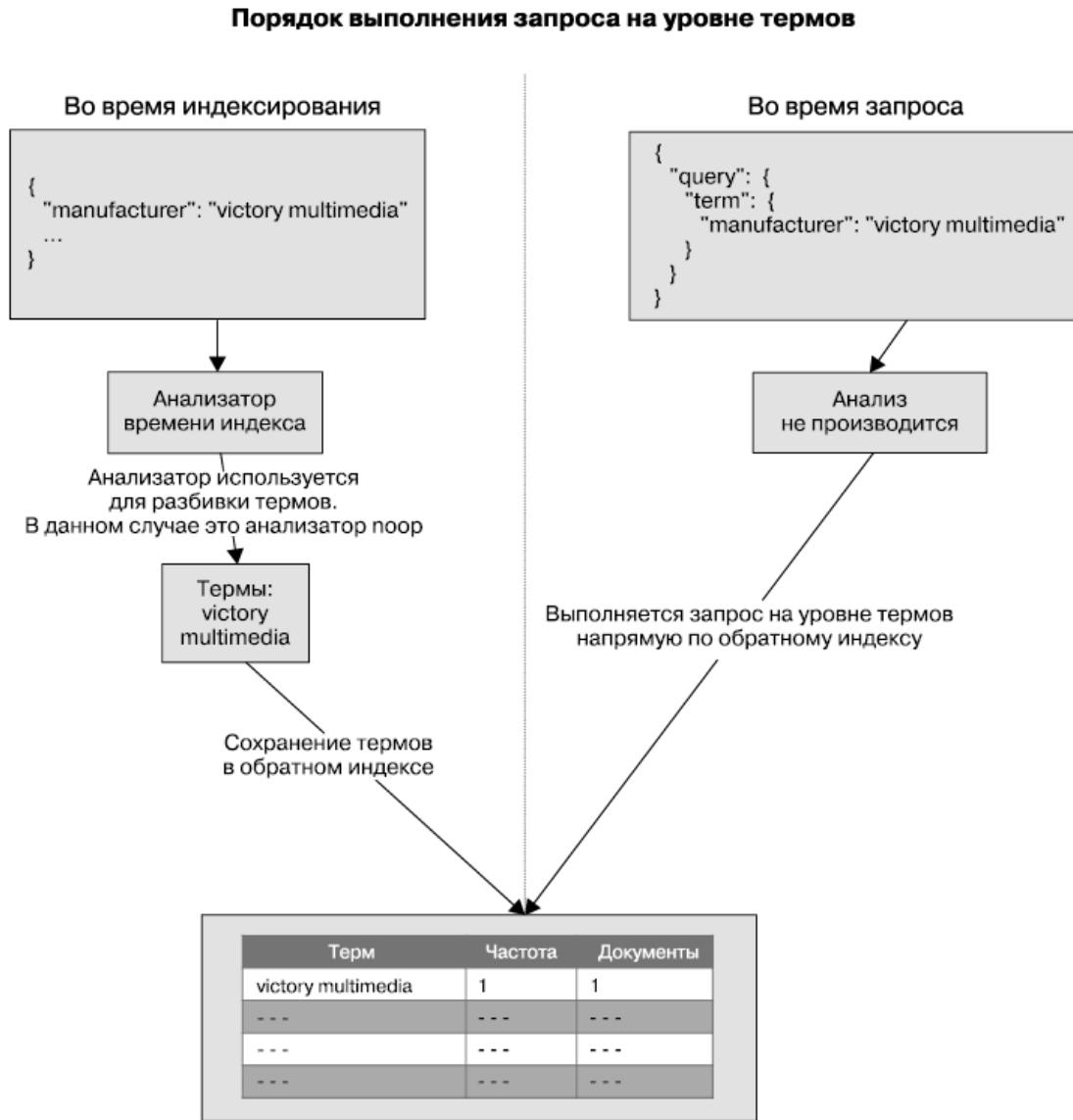


Рис. 3.2

Как видите, рисунок разделен на две части. Левая половина показывает происходящее во время индексирования, а в правой вы видите, что происходит во время выполнения запроса на уровне термов.

Взглянем на левую часть рисунка — на происходящее во время индексирования. В частности, мы видим построение обратного индекса и создание запроса для поля `manufacturer.raw`. Вспомним, что согласно нашему определению индекса поле `manufacturer.raw` имеет тип `keyword`. Поля этого типа не анализируются; значение поля сохраняется напрямую как терм в обратном индексе.

В правой части рисунка показано, что происходит, когда мы ищем по запросу `term`, который является запросом на уровне термов. Запрос `term`, как мы увидим в этом разделе позже, напрямую переходит к поиску терма `victor multimedia`, не разбивая его и не используя анализатор. Такие запросы полностью пропускают процесс анализа и ищут необходимый терм напрямую в обратном индексе.

Запросы на уровне термов создают слой-основу, на котором строятся другие, высокоуровневые полнотекстовые запросы. Их мы рассмотрим в следующем разделе.

Мы изучим такие типы запросов:

- запрос диапазона;
- запрос существования;
- `term`-запрос (термин-запрос);
- запрос терминов.

Запрос диапазона

Запросы диапазона (`range query`) могут применяться к полям с типами данных, которые естественно упорядочены. Например, к normally упорядоченным относятся целые числа и даты. Не составляет труда выяснить, какое значение меньше, равно или

больше других значений. Благодаря нормальному порядку, свойственному этим типам данных, мы можем выполнять по отношению к ним запрос диапазона.

Мы обсудим, как применять запросы диапазона в следующих случаях:

- для числовых типов;
- с увеличением оценки;
- для дат.

Взглянем на типичный запрос диапазона для числового поля.

Запрос диапазона для числового типа

Предположим, что мы храним продукты с их ценами в индексе Elasticsearch и хотим выбрать все продукты в пределах определенного диапазона. Запрос для выбора продуктов в диапазоне от \$10 до 20 выглядит следующим образом:

```
GET /amazon_products/products/_search
{
  "query": {
    "range": {
      "price": {
        "gte": 10,
        "lte": 20
      }
    }
  }
}
```

Ответ на такой запрос выглядит так:

```
{  
    "took": 1,  
    "timed_out": false,  
    "_shards": {  
        "total": 1,  
        "successful": 1,  
        "failed": 0  
    },  
    "hits": {  
        "total":  
201,  
        "max_score":  
1,  
        "hits": [  
            {  
                "_index": "amazon_products",  
                "_type": "products",  
                "_id": "AV5lK4WiaMctupbz_61a",  
                "_score":  
1,  
                "_source": {  
                    "price":  
"19.99",  
                    "description": "reel deal casino  
championship edition  
(win 98 me nt 2000  
xp)",  
                    "id": "b00070ouja",  
                    "title": "reel deal casino championship  
edition",  
                }  
            }  
        ]  
    }  
}
```

```
        "manufacturer": "phantom efx",
        "tags": []
    },
},
```

Обратите внимание на следующее.

1. В поле `hits.total` мы видим, сколько всего результатов было найдено. В данном случае мы имеем 201 соответствие условиям поиска.
2. Поле `hits.max_score` показывает оценку документа, который лучше всех соответствует запросу. Поскольку запрос диапазона является структурированным и не учитывает релевантность, он работает как фильтр. Он не меняет оценку документа. Все документы имеют оценку 1.
3. Массив `hits.hits` представляет собой список всех фактических соответствий. По умолчанию Elasticsearch не возвращает 201 результат в один заход. Возвращаются только первые десять записей. Если вы хотите увидеть все результаты, можете это легко сделать с помощью нескольких запросов, которые мы рассмотрим позже.
4. Поле `price` во всех результатах поиска будет в пределах запрошенного диапазона, то есть $10 \leq price \leq 20$.

Запрос диапазона с увеличением оценки

По умолчанию запрос диапазона присваивает оценку 1 каждому соответствующему документу. Но допустим, вы используете запрос диапазона вместе с каким-нибудь другим запросом и хотите присвоить более высокую оценку полученным документам, если они удовлетворяют тем или

иным условиям. Можно использовать составные запросы, такие как булев, где допустимо совмещать разные типы запросов. Запрос диапазона позволяет установить параметр **boost** для изменения оценки документа в зависимости от условий других запросов:

```
GET /amazon_products/products/_search
{
  "from": 0,
  "size": 10,
  "query": {
    "range": {
      "price": {
        "gte": 10,
        "lte": 20,
        "boost": 2.2
      }
    }
  }
}
```

Все документы, которые пройдут фильтр, в этом запросе получат оценку 2.2 вместо 1.

Запрос диапазона для дат

Вы можете выполнить запрос диапазона для полей с датами, так как они также имеют естественный порядок. Во время такого запроса можно указать формат отображения данных:

```
GET /orders/order/_search
{
  "query": {
```

```
"range" : {  
    "orderDate" : {  
        "gte": "01/09/2017",  
        "lte": "30/09/2017",  
        "format": "dd/MM/yyyy"  
    }  
}  
}  
}  
}
```

Такой запрос отфильтрует все заказы, сделанные в сентябре 2017 года.

Elasticsearch позволяет использовать в запросах даты с указанием или без указания времени. Поддерживается также использование особых условий, включая `now` для отметки текущего времени. Например, в коде далее запрашиваются данные за последние семь дней до сегодняшнего, а именно данные за 24×7 часов до текущего момента с точностью до миллисекунд:

```
GET /orders/order/_search  
{  
    "query": {  
        "range" : {  
            "orderDate" : {  
                "gte": "now-7d",  
                "lte": "now"  
            }  
        }  
    }  
}
```

Возможность использовать условия типа now облегчает охват при поиске.



Elasticsearch поддерживает много математических операций с датами. Для дат возможно также использование ключевого слова now. Кроме того, поддерживается добавление или вычитание времени с различными единицами измерения. Вы можете использовать сокращения до одного символа, такие как у (год), М (месяц), w (неделя), d (день), h или Н (часы), т (минуты) и s (секунды). Например: now - 1у будет означать выбор времени длительностью один год до текущего момента. Время можно округлять в разных единицах. Например, чтобы округлить интервал до дня, включая оба начальных и конечных промежутка, используйте выражение "gte": "now - 7d/d" или "lte": "now/d". Добавляя /d, вы округлите время до дней.

Запрос диапазона работает в контексте фильтра по умолчанию. Он не вычисляет оценки и всегда присваивает оценку 1 для всех соответствующих документов.

Запрос существования

Иногда полезно получить только те записи, которые имеют не нулевые и не пустые значения в конкретных полях. Например, так выполняется выбор всех полей с заполненным описанием:

```
GET /amazon_products/products/_search
{
  "query": {
```

```
"exists": {  
    "field": "description"  
}  
}  
}
```

Запрос существования делает из запроса фильтр; другими словами, он работает в *контексте фильтра*. Его работа схожа с действием запроса диапазона, когда оценки не имеют значения.



Что такое контекст фильтра? Когда запрос выполняется только для фильтрации документов, а именно для решения о включении конкретного документа в результаты, важно пропустить процесс оценки. Elasticsearch может пропускать этот процесс для определенных типов запросов и присваивать универсальную оценку 1 каждому документу, который соответствует критериям фильтрации. Это не только ускоряет выполнение запроса (благодаря пропуску процесса оценки), но и позволяет Elasticsearch кэшировать результаты фильтров. В Elasticsearch результаты фильтрации кэшируются путем обслуживания массивов единиц и нулей.

Term-запрос

Как найти все продукты от конкретного производителя? Мы знаем, что в наших данных поле производителя имеет строковый тип. Название производителя может содержать пробелы. В данный момент нам нужен точный поиск. Например, когда мы ищем `victory multimedia`, нам не

нужны результаты, содержащие только `victory` или только `multimedia`. Для выполнения такого поиска мы можем использовать term-запрос, или термин-запрос (term query).

Определяя поле производителя, мы сохраняем его с типом `text` или `keyword`. Для точного запроса нам нужно использовать поле типа `keyword`:

```
GET /amazon_products/products/_search
{
  "query": {
    "term": {
      "manufacturer.raw": "victory multimedia"
    }
  }
}
```

Term-запрос является низкоуровневым в том понимании, что он не выполняет никакого анализа термина. Он также напрямую работает с обратным индексом, который сформирован из указанных полей, в данном случае это поле `manufacturer.raw`. По умолчанию термин-запрос работает в контексте запроса и, следовательно, вычисляет оценку.

Ответ на запрос выглядит следующим образом (вы видите только его часть):

```
{
  ...
  "hits": {
    "total": 3,
    "max_score": 5.965414,
    "hits": [
      {
```

```
"_index": "amazon_products",
"_type": "products",
"_id": "AV5rBfPNNI_2eZGciIHc",
"_score": 5.965414,
...

```

Как видите, каждому документу дана оценка по умолчанию. Для выполнения термин-запроса в контексте фильтра без оценки его нужно запустить внутри фильтра `constant_score`.

Теперь этот запрос вернет одинаковые результаты для всех соответствующих документов. Запрос `constant_score` мы еще рассмотрим более детально далее в этой главе. На текущий момент вы можете считать, что он делает оценивающий запрос неоценивающим. Во всех случаях, когда нам не нужно знать, насколько хорошо документ соответствует запросу, мы можем ускорить работу запроса, используя `constant_score` с параметром `filter`. Есть и другие типы составных запросов, которые помогут нам конвертировать разные типы запросов и комбинировать их с другими. Мы подробно рассмотрим их во время изучения сложных составных запросов.

Полнотекстовый поиск

Полнотекстовые запросы могут работать с неструктуризованными текстовыми полями. Эти запросы знают о процессе анализа и применяют анализатор к полям до того, как фактически начинать процесс поиска. Сначала проводится подбор корректного анализатора путем проверки определения `search_analyzer`, а также наличия анализатора, в обоих случаях — на уровне поля. Если анализаторы на уровне поля не выявлены, предпринимается попытка использовать анализатор, определенный на уровне индекса.

Таким образом, полнотекстовые запросы выполняют необходимые процессы анализа полей до осуществления фактических запросов. Их также можно называть *высокоуровневыми запросами*. Разберемся в структуре выполнения высокоровневого запроса (рис. 3.3).

Здесь мы можем видеть, как будет выполнен один высокоровневый запрос для поля названия. Вспомните, что согласно определению нашего индекса тип поля названия — `text`. Во время индексирования значение поля обрабатывается анализатором. В данном случае был применен стандартный анализатор, следовательно, обратный индекс был разбит на термины, такие как `god`, `heroes`, `rome` и т.д.

Во время запроса (обратите внимание на правую часть рисунка) мы выполняем высокоровневый запрос `match`. Более детально о запросе соответствия мы еще поговорим далее в разделе; это один из возможных высокоровневых запросов. Поисковые термины обрабатываются запросом `match` и анализируются стандартным анализатором. После его применения отдельные термины далее используются для отдельных запросов на уровне термов.

Порядок выполнения высокогоревневого запроса

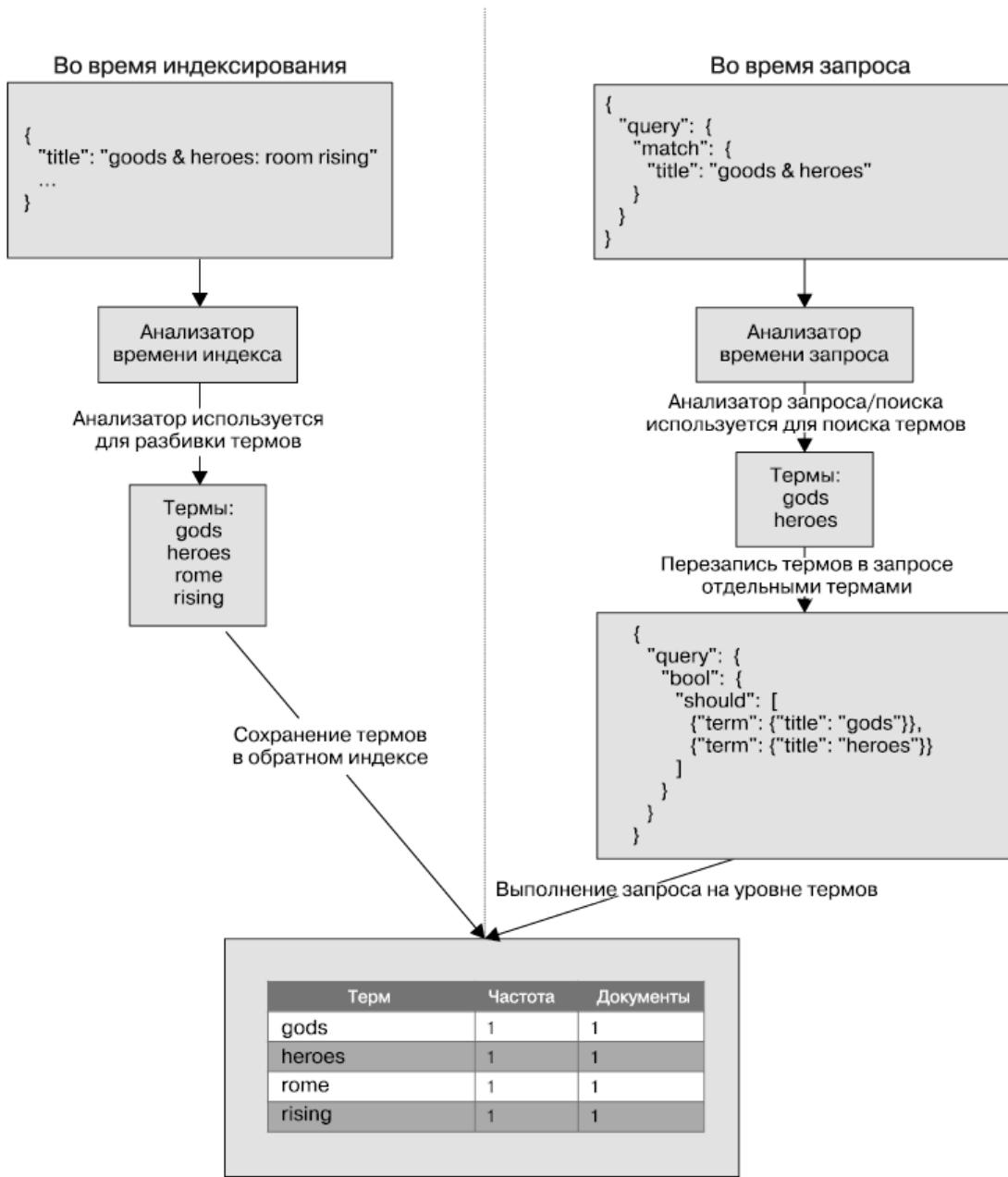


Рис. 3.3

В данном примере результатом будет несколько запросов на уровне термов — по одному на каждый терм после применения анализатора. Исходным поисковым запросом было выражение `goods heroes`, в результате было получено два терма — `goods` и `heroes`, которые используются как отдельные термы со своими

запросами. Далее эти два запроса комбинируются с помощью составного запроса `bool`. Подробнее о составных запросах мы поговорим в следующем разделе.

Далее мы рассмотрим полнотекстовые запросы:

- соответствия;
- соответствия фразы;
- нескольких соответствий.

Запрос соответствия

Запрос соответствия (`match query`) — это запрос, который по умолчанию применяется в большинстве случаев полнотекстового поиска. Это один из высокоуровневых запросов, который знает об использовании анализатора для исходного поля. Посмотрим, как это работает.

Например, когда вы применяете запрос соответствия для поля типа `keyword`, он знает о наличии типа `keyword` у поля и, следовательно, во время запроса поисковые термины не анализируются:

```
GET /amazon_products/products/_search
{
  "query": {
    "match": {
      "manufacturer.raw": "victory multimedia"
    }
  }
}
```

В данном случае запрос соответствия работает так же, как термин-запрос, о котором мы говорили в предыдущем разделе. Он не анализирует элемент `victory multimedia` как раздельные термины `victory` и `multimedia`. Это связано с тем, что мы выполняем запрос в отношении поля `keyword`, а именно — `manufacturer.raw`. Фактически в этом конкретном случае запрос соответствия конвертируется в термин-запрос, как видно ниже:

```
GET /amazon_products/products/_search
{
  "query": {
    "term": {
      "manufacturer.raw": "victory multimedia"
    }
  }
}
```

В таком случае термин-запрос возвращает те же оценки, что и запрос соответствия, так как оба выполнены для поля `keyword`.

Теперь посмотрим, что произойдет, если выполнить запрос соответствия в отношении поля `text`, что является типичным примером полнотекстового запроса:

```
GET /amazon_products/products/_search
{
  "query": {
    "match": {
      "manufacturer": "victory multimedia"
    }
  }
}
```

}

Когда мы выполняем запрос соответствия, мы ожидаем от него следующих действий.

1. Искать термины `victory multimedia` по всем документам в пределах поля производителя.
2. Найти лучшие соответствия, отсортированные по оценке в порядке убывания.
3. Если оба термина найдены рядом и в том же порядке, документ должен получить более высокую оценку, чем другие документы, в которых есть совпадение по обоим терминам в том же порядке, но не рядом друг с другом.
4. Включать в результаты поиска документы, в которых есть либо `victory`, либо `multimedia`, но присваивать им более низкую оценку.

Запрос соответствия с параметрами по умолчанию может выполнить все перечисленные шаги и найти лучшие соответствующие документы, а также отсортировать их согласно оценке (от высокой к низкой).

Работа запроса соответствия по умолчанию предусматривает наличие только поисковых терминов. Но вы можете указать для него дополнительные параметры. К стандартным параметрам относятся следующие:

- оператор;
- параметр `minimum_should_match`;
- параметр `fuzziness` (нечеткости).

Оператор

По умолчанию, если поисковый термин после работы анализатора разбивается на несколько термов, требуется способ скомбинировать результаты из отдельных терминов. Как вы уже убедились на предыдущем примере, стандартное поведение запроса `match` состоит в комбинировании результатов с помощью оператора `or`, то есть один из терминов должен присутствовать в поле документа.

Вы можете изменить оператор на `and` с помощью следующего запроса:

```
GET /amazon_products/products/_search
{
  "query": {
    "match": {
      "manufacturer": {
        "query": "victory multimedia",
        "operator": "and"
      }
    }
  }
}
```

В таком случае в поле производителя в документе должны присутствовать оба термина: и `victory`, и `multimedia`.

minimum_should_match

Мы можем не менять оператор с `or` на `and` и хотя бы указать, сколько терминов должно быть найдено в документе для того, чтобы он был включен в результаты поиска. Таким образом вы можете контролировать точность результатов поиска:

```
GET /amazon_products/products/_search
{
  "query": {
    "match": {
      "manufacturer": {
        "query": "victory multimedia",
        "minimum_should_match": 2
      }
    }
  }
}
```

Запрос выше работает схожим образом, как при использовании оператора `and`, поскольку в нем есть два термина и мы указали, что минимальное количество совпадений — два.

С помощью параметра `minimum_should_match` мы можем указать минимальное количество совпадений в документе.

Нечеткость

С параметром `fuzziness` мы можем превратить запрос соответствия в запрос нечеткости. Его работа основана на расстоянии Левенштейна¹ для преобразования одного поискового термина в другой путем выполнения нескольких правок исходного текста. Под правками мы подразумеваем вставки, удаления, замещения или изменения мест символов в оригинальном термине. Параметр `fuzziness` может иметь одно из следующих значений: 0, 1, 2 или AUTO.

Например, в следующем запросе есть опечатка: `victor` вместо `victory`. Используя параметр `fuzziness` со значением

1, мы все равно найдем все записи касательно `victory multimedia`:

```
GET /amazon_products/products/_search
{
  "query": {
    "match": {
      "manufacturer": {
        "query": "victor multimedia",
        "fuzziness": 1
      }
    }
  }
}
```

Если мы хотим оставить больше маневров для исправления ошибок, то увеличим значение нечеткости до 2. При таком значении параметра `fuzziness` возможен поиск даже по запросу `victer`, поскольку это слово преобразуется в `victory` двумя исправлениями:

```
GET /amazon_products/products/_search
{
  "query": {
    "match": {
      "manufacturer": {
        "query": "victer multimedia",
        "fuzziness": 2
      }
    }
  }
}
```

Выбор AUTO будет автоматически подбирать числовое значение нечеткости из 0, 1, 2 в зависимости от длины исходного термина. При автоматической настройке термины длиной до двух символов будут иметь допустимую нечеткость 0 (должно быть четкое соответствие), термины от трех до пяти символов будут иметь нечеткость 1, а термины с пятью и более символами — нечеткость, равную 2.

Нечеткость имеет свою цену, так как Elasticsearch обязан создавать дополнительные термы для соответствий. Поддерживаются следующие параметры для контролирования количества термов:

- `max_expansions` — максимальное количество термов после расширения;
- `prefix_length` — число, например 0, 1, 2 и т.д. Задает размер префикса, помогает отсеять количество слов при поиске.

Запрос соответствия фразы

Запрос соответствия фразы может быть полезен, когда требуется найти сочетание слов, а не отдельные термины в документе.

Например, следующий текст представлен как часть описания к одному из продуктов:

```
real video saltware aquarium on your desktop!
```

Нам нужно найти все продукты, в описании которых есть именно такое сочетание слов: `real video saltware aquarium`. Для получения необходимого результата мы можем использовать запрос `match_phrase`. Запрос `match` не

сработает, так как он не учитывает сочетание терминов и их близость друг к другу. Такой запрос попросту включит в результаты все документы, в которых встречается какое-либо слово из перечисленных, даже если они не имеют указанного порядка в документе:

```
GET /amazon_products/products/_search
{
  "query": {
    "match_phrase": {
      "description": {
        "query": "real video saltware aquarium"
      }
    }
  }
}
```

Ответ будет выглядеть следующим образом:

```
{
  ...
  "hits": {
    "total": 1,
    "max_score": 22.338196,
    "hits": [
      {
        "_index": "amazon_products",
        "_type": "products",
        "_id": "AV5rBfasNI_2eZGciIbg",
        "_score": 22.338196,
        "_source": {
          "price": "19.95",
```

```

        "description": "real video saltware
aquarium on your
    desktop! product information see real
fish swimming on your
    desktop in fullmotion video! you'll
find exotic saltwater
    fish such as sharks angelfish and
more! enjoy the beauty
    and serenity of a real aquarium at
yourdesk",
        "id": "b00004t2un",
        "title": "sales skills 2.0 ages 10+",
        "manufacturer": "victory multimedia",
        "tags": []
    }
}
]
}
}

```

Запрос `match_phrase` также поддерживает параметр `slop`, который позволяет указать целое число: 0, 1, 2, 3 и т.д. `slop` уменьшает количество слов/терминов, которые могут быть пропущены во время запроса.

Например, при значении `slop`, равном 1, даже при одном отсутствующем слове из всей фразы документ все равно будет обозначен как соответствующий поиску:

```

GET /amazon_products/products/_search
{
    "query": {
        "match_phrase": {

```

```
        "description": {  
            "query": "real video aquarium",  
            "slop": 1  
        }  
    }  
}
```

Таким образом, `slop 1` разрешит пользователю ввести для поиска `real video aquarium` или `real saltware aquarium` и все равно найти документ, который содержит полную фразу `real video saltware aquarium`. Значение `slop` по умолчанию равно `0`.

Запрос нескольких соответствий

Запрос нескольких соответствий представляет собой расширение обычного запроса соответствия. Он позволяет нам запускать запрос `match` по нескольким полям, а также имеет несколько вариаций для подсчета общей оценки документов.

Вы можете использовать запрос нескольких соответствий с различными параметрами. Мы рассмотрим следующие виды этого запроса:

- запрос по нескольким полям по умолчанию;
- запрос с увеличением оценки одного или нескольких полей.

Запрос по нескольким полям по умолчанию

Мы хотим предоставить своему веб-приложению функционал поиска по продуктам. Когда конечный пользователь ищет какие-либо термины, нам требуется выполнять запросы по

обоим полям *названия* и *описания*. Это возможно с помощью запроса по нескольким полям.

Следующий запрос найдет все документы, которые имеют слова `monitor` или `aquarium` в полях названия или описания:

```
GET /amazon_products/products/_search
{
  "query": {
    "multi_match": {
      "query": "monitor aquarium",
      "fields": ["title", "description"]
    }
  }
}
```

Такой запрос уделяет одинаковое внимание обоим полям. Рассмотрим, как повысить оценку одного или нескольких полей.

Увеличение оценки одного или нескольких полей

В веб-приложениях для электронной коммерции нередко бывает так, что пользователь ищет какие-либо товары, используя некоторые ключевые слова. Что, если мы хотим сделать поле *название* более важным, чем поле *описания*? Если одно или несколько ключевых слов совпадет в названии, это будет более релевантный продукт, чем тот, который имеет совпадения только в описании. Можно увеличить оценку документа в тех случаях, если совпадение найдено в определенных полях.

Сделаем поле *название* в три раза более важным, чем поле *описания*. Для этого можно использовать следующий синтаксис:

```
GET /amazon_products/products/_search
{
  "query": {
    "multi_match": {
      "query": "monitor aquarium",
      "fields": ["title3", "description"]
    }
  }
}
```

Запрос нескольких соответствий предлагает больше контроля для комбинирования оценок из разных полей.

В этом разделе мы поговорили о полнотекстовых запросах, известных также как высокоуровневые запросы. Эти запросы находят лучшие *соответствующие* документы согласно оценке. Высокоуровневые запросы на внутреннем уровне используют некоторые запросы на уровне термов. В следующем разделе вы узнаете, как писать сложные составные запросы.

Написание составных запросов

Запросы этого типа можно использовать при необходимости совмещения одного или нескольких запросов. Некоторые составные запросы конвертируют оценивающие запросы в неоценивающие либо комбинируют несколько оценивающих и неоценивающих запросов. Мы рассмотрим следующие типы составных запросов:

- запрос постоянной оценки;
- булев запрос.

Запрос постоянной оценки

Elasticsearch поддерживает структурированные и полнотекстовые запросы. Первые не нуждаются в механизме оценок, в то время как вторым необходимы оценки для поиска лучшего *соответствия* среди документов. Запрос постоянной оценки позволяет конвертировать запрос оценки, который обычно работает в контексте запроса, для выполнения в контексте неоценивающего фильтра. Запрос постоянной оценки наверняка займет важное место среди вашего инструментария.

Например, термин-запрос обычно работает в контексте запроса. Это значит, что при выполнении такого запроса Elasticsearch не только фильтрует документы, но и оценивает их:

```
GET /amazon_products/products/_search
{
  "query": {
    "term": {
      "manufacturer.raw": "victory multimedia"
    }
  }
}
```

Обратите внимание на выделенный код. Этот фрагмент фактически и есть термин-запрос. По умолчанию выделенный JSON-элемент `query` определяет контекст запроса.

Ответ содержит оценку для каждого документа. Рассмотрите следующую часть ответа:

```
{
  . . . ,
```

```
"hits": {
    "total": 3,
    "max_score": 5.966147,
    "hits": [
        {
            "_index": "amazon_products",
            "_type": "products",
            "_id": "AV5rBfasNI_2eZGciIbg",
            "_score": 5.966147,
            "_source": {
                "price": "19.95",
            ...
        }
    ]
}
```

Здесь мы намеревались фильтровать документы, следовательно, не было необходимости вычислять оценку релевантности каждого документа.

Оригинальный запрос может быть конвертирован для работы в контексте фильтра, для этого используем следующий составной запрос оценки:

```
GET /amazon_products/products/_search
{
    "query": {
        "constant_score": {
            "filter": {
                "term": {
                    "manufacturer.raw": "victory
multimedia"
                }
            }
        }
    }
}
```

```
    }
}
```

Как видите, мы обернули оригиналый выделенный объект `term` и его дочерний объект. Таким образом мы присваиваем нейтральную оценку 1 каждому документу по умолчанию. Рассмотрите часть ответа в следующем коде:

```
{
  ...
  "hits": {
    "total": 3,
    "max_score_score
```

Можно указать параметр `boost`, который присвоит эту оценку вместо нейтральной:

```
GET /amazon_products/products/_search
{
  "query": {
```

```
"constant_score": {  
    "filter": {  
        "term": {  
            "manufacturer.raw": "victory  
multimedia"  
        }  
    },  
    "boost": 1.2  
}  
}  
}
```

В чем толк от увеличения оценки каждого документа до 1.2 этим фильтром? Толка нет, если этот запрос используется сам по себе. Но если его комбинировать с другими запросами, применяя, например, запрос `bool`, увеличенная оценка обретает смысл. Все документы, прошедшие этот фильтр, будут иметь более высокую оценку по сравнению с другими документами, комбинированными с другими запросами.

Булев запрос

Булев запрос в Elasticsearch — ваш швейцарский нож. Он поможет вам с написанием многих видов сложных запросов. И если у вас есть опыт работы с SQL, вы уже знаете, как работают фильтры с операторами AND и OR в условии WHERE. Булев запрос разрешает вам комбинировать несколько оценивающих и неоценивающих запросов.

Сначала рассмотрим, как внедрить простые операторы AND и OR.

Булев запрос имеет следующие разделы:

```

GET /amazon_products/products/_search
{
  "query": {
    "bool": {
      "must": [...],           scoring queries
      executed in query context
      "should": [...],        scoring queries
      executed in query context
      "filter": {},           non-scoring queries
      executed in filter context
      "must_not": [...]      non-scoring queries
      executed in filter context
    }
  }
}

```

Запросы с условиями `must` и `should` выполняются в контексте запроса, кроме тех случаев, когда весь булев запрос включен в контексте фильтра.

Фильтр и запросы `must_not` всегда выполняются в контексте фильтра. Они всегда будут возвращать нулевую оценку и обеспечивать фильтрацию документов.

Разберемся, как сформировать неоценивающий запрос, который занимается исключительно неструктурированным поиском. Мы рассмотрим, как формулировать следующие типы структурированных поисковых запросов с помощью булева запроса:

- комбинирование условий OR;
- комбинирование условий AND и OR;
- добавление условий NOT.

Комбинирование условий OR

Далее приведен код, позволяющий найти все продукты в ценовом диапазоне от 10 до 13 или (OR) произведенные компанией valuesoft:

```
GET /amazon_products/products/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "should": [
            {
              "range": {
                "price": {
                  "gte": 10,
                  "lte": 13
                }
              }
            },
            {
              "term": {
                "manufacturer.raw": {
                  "value": "valuesoft"
                }
              }
            }
          ]
        }
      }
    }
  }
}
```

```
    }
}
```

Поскольку мы применяем условие OR, мы поместили его под строкой `should`. Поскольку нам не нужны оценки, мы обернули наш запрос `bool` в запрос постоянной оценки.

Комбинирование условий AND и OR

Следующий код позволяет найти все продукты в ценовом диапазоне от 10 до 13 и (AND) произведенные компанией Valuesoft или Pinnacle:

```
GET /amazon_products/products/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "must": [
            {
              "range": {
                "price": {
                  "gte": 10,
                  "lte": 30
                }
              }
            }
          ],
          "should": [
            {
              "term": {

```

Обратите внимание, что все условия, для которых необходим союз OR, размещены внутри `should`. А условия, относящиеся к AND, находятся в элементе `must`. Тем не менее вы можете поместить все условия, которым необходим оператор AND, в элемент `filter`.

Добавление условий NOT

Мы можем добавить оператор NOT, например, для того, чтобы отфильтровать определенные элементы, используя условие `must_not` в фильтре `bool`.

Найдем все продукты стоимостью от 10 до 20, которые не произведены компанией encore. Для этого выполним

следующий запрос:

```
GET /amazon_products/products/_search
{
  "query": {
    "constant_score": {
      "filter": {
        "bool": {
          "must": [
            {
              "range": {
                "price": {
                  "gte": 10,
                  "lte": 20
                }
              }
            }
          ],
          "must_not": [
            {
              "term": {
                "manufacturer.raw": "encore"
              }
            }
          ]
        }
      }
    }
  }
}
```

Запрос `bool` с элементом `must_not` полезен для отрицания любого запроса. Для применения фильтра NOT к запросу он должен быть обернут в булев запрос в строке `must_not`, как показано ниже:

```
GET /amazon_products/products/_search
{
  "query": {
    "bool": {
      "must_not": {
        ... оригинальный запрос для выполнения
        операции NOT...
      }
    }
  }
}
```

Обратите внимание, что не требуется обрамлять весь запрос в запрос постоянной оценки, если мы используем только `must_not` для отрицания запроса. Запрос `must_not` всегда выполняется в контексте фильтра.

Итак, мы рассмотрели различные типы сложных составных запросов. Elasticsearch поддерживает еще больше таких запросов, включая следующие:

- запрос Dis Max;
- запрос оценки функции;
- запрос увеличения оценки;
- запрос индексов.

В данной книге мы не будем рассматривать эти виды запросов. При необходимости вы можете сами изучить их, ведь у вас уже есть базовое понимание составных запросов, используемых в Elasticsearch. Для этого обратитесь к документации Elasticsearch.

Резюме

В этой главе мы еще подробнее рассмотрели возможности поиска Elasticsearch. Мы разобрались, в чем значение анализаторов и из чего они состоят. Поговорили о том, как использовать некоторые из встроенных анализаторов Elasticsearch, как создавать свои собственные нестандартные анализаторы.

На текущий момент вы изучили два основных типа запросов: запросы на уровне термов и полнотекстовые запросы. Кроме того, научились создавать более сложные, составные запросы.

В этой главе вы получили основные сведения, на которые можете опираться при работе с запросами данных в Elasticsearch. Есть много других типов запросов, поддерживаемых Elasticsearch, но мы рассмотрели лишь самые важные из них. Благодаря этим знаниям вы уже сейчас можете начать работу, а разобраться в других типах запросов вам поможет документация Elasticsearch.

В главе 4 рассказывается о возможностях аналитики, предоставляемых данной программой. Мы изучим основной компонент Elastic Stack и Elasticsearch, а также другие полезные инструменты.

¹ Расстояние Левенштейна между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой,

необходимых для превращения одной строки в другую («Википедия»). — *Здесь и далее примеч. ред.*

4. Анализ данных с помощью Elasticsearch

На текущем этапе изучения Elastic Stack 6.0 вы уже должны были хорошо разобраться в работе Elasticsearch. В предыдущих двух главах мы рассмотрели фундаментальные основы этой программы, а также реальные примеры ее практического применения.

Технология в основе Elasticsearch — Apache Lucene — изначально создавалась для текстового поиска. Благодаря инновациям в разработке этих двух программ они зарекомендовали себя как мощный поисковый движок. В этой главе мы поговорим о том, что именно делает Elasticsearch хорошей поисковой системой. Мы рассмотрим следующие темы.

- Основы агрегации.
- Подготовка данных к анализу.
- Метрическая агрегация.
- Сегментарная агрегация.
- Агрегация контейнеров.

Все это мы изучим на реальном наборе данных. Начнем с основ агрегации.

Основы агрегации

В отличие от поиска аналитика предназначена для получения куда более полной картины. При поиске мы детально

рассматриваем несколько записей; аналитика же позволяет взглянуть на данные шире и разбить их различными способами. Когда мы изучали поиск, мы использовали API следующего вида:

```
POST /<index_name>/<type_name>/_search
{
  "query": {
    ...
    ... тип запроса ...
  }
}
```

Все запросы агрегации имеют схожий вид. Разберемся в структуре.

Агрегация, или элемент `aggs`, позволяет нам агрегировать данные. Все такие запросы имеют следующий вид:

```
POST /<index_name>/<type_name>/_search
{
  "aggs": {
    ...
    ... тип агрегации ...
  },
  "query": { ... тип запроса ... }, // необязательная часть запроса
  "size": 0                         // обычно указывается размер 0
}
```

Элемент `aggs` должен содержать фактический запрос агрегации. Тело запроса зависит от желаемого типа агрегации. Необязательный элемент `query` указывает контекст агрегации. Агрегация будет учитывать все документы данного индекса и

типа, если не указан элемент `query` (мы можем считать его равным запросу `match_all`, если нет иного запроса). Если мы хотим ограничить контекст агрегации, необходимо указать элемент `query`. Например, мы хотим, чтобы агрегация работала не по всем данным, а только по определенным документам, которые соответствуют конкретным условиям. Этот запрос фильтрует документы, которые будут обработаны запросом `aggs`.

Элемент `size` указывает, сколько соответствий поиска должно быть возвращено в ответе. Значение по умолчанию составляет 10. Если значение `size` не указано, ответ будет содержать десять соответствий из контекста запроса. Обычно, если мы заинтересованы только в получении результатов агрегации, необходимо присваивать элементу `size` значение 0, чтобы не получить иные результаты.

В широком понимании Elasticsearch поддерживает четыре типа агрегаций:

- сегментарные агрегации;
- метрические агрегации;
- матричные агрегации;
- агрегации контейнеров.

Сегментарные агрегации

Сегментарные агрегации разделяют данные запроса (обозначенные контекстом `query`) на различные сегменты, идентифицируемые сегментарным ключом. Такой тип агрегации оценивает документ в контексте путем вычисления, в какой именно сегмент он попадает. В результате мы

получаем набор определенных сегментов со своими ключами и документами.

Люди, работающие с SQL, для этого используют запрос с оператором GROUP BY, например:

```
SELECT column1, count(*) FROM table1 GROUP BY column1;
```

Данный запрос разделяет таблицу по разным значениям в столбце 1 и возвращает количество документов в пределах каждого значения столбца 1. Это пример сегментарной агрегации. Elasticsearch поддерживает много разных типов сегментарной агрегации, мы еще рассмотрим их в этой главе.

Сегментарные агрегации могут быть указаны в самом верху или на внешнем уровне запроса агрегации. Их также можно добавлять в другие агрегации такого типа.

Метрические агрегации

Метрические агрегации работают с полями числовых типов. Они вычисляют ценность числового поля в конкретном контексте. Например, у нас есть таблица с результатами студенческих экзаменов. Каждая запись хранит отметки, полученные студентами. Метрическая агрегация позволяет вычислить разные сводные показатели этого столбца с отметками, например сумму, среднее, минимальное, максимальное значение и т.д.

В терминах SQL следующий запрос будет грубой аналогией того, что может сделать метрическая агрегация:

```
SELECT avg(score) FROM results;
```

Этот запрос считает среднюю отметку в данном контексте. В этом случае контекстом является вся таблица, то есть все

студенты. Это пример метрической агрегации.

Метрические агрегации можно размещать в самом верху или на внешнем уровне запросов агрегаций. Их также можно добавлять в другие агрегации сегментарного типа. Но в метрические агрегации нельзя вставлять агрегации других типов.

Матричные агрегации

Матричные агрегации впервые были представлены в версии Elasticsearch 5.0. Они работают с несколькими полями и вычисляют матрицы по всем документам в пределах контекста запроса.

Матричные агрегации могут быть вставлены в сегментарные, но обратная вставка невозможна. Это все еще относительно новая функция. Детальное изучение матричных агрегаций не входит в задачи этой книги.

Агрегации контейнеров

Агрегации контейнеров являются агрегациями высшего порядка и могут использовать вывод других агрегаций. Это может быть полезным для вычисления чего-либо, например производных. Более подробно агрегации контейнеров мы рассмотрим далее в этой главе.

Итак, это был краткий обзор различных типов агрегаций, поддерживаемых в Elasticsearch. Агрегации контейнеров и агрегации матричного типа являются относительно новыми и применяются не так широко, как метрические или сегментарные. Оба этих типа мы рассмотрим более детально далее в этой главе.

В следующем разделе мы загрузим и подготовим данные, чтобы лучше разобраться в работе агрегаций, рассматриваемых в этой главе.

Подготовка данных к анализу

В качестве примера мы используем данные сетевого трафика, собранные с Wi-Fi-роутеров. На протяжении всей главы мы будем анализировать данные из этого примера. Важно понять, как выглядят записи рассматриваемой системы и что они обозначают. Мы рассмотрим следующие темы:

- структуру данных;
- загрузку данных с помощью Logstash.

Кроме того, подготовим и загрузим данные в локальный процесс Elasticsearch.

Структура данных. На рис. 4.1 показано устройство системы сбора данных с роутеров. Данные о сетевом трафике и использовании пропускной способности сети собираются, после чего поступают на хранение в Elasticsearch.

Эти данные собраны системой для следующих целей.

- В левой части рисунка вы видите несколько квадратов, которые обозначают клиентское оборудование: маршрутизаторы Wi-Fi, развернутые на месте, а также все устройства, подключенные к этим беспроводным маршрутизаторам. Среди подключенных устройств есть ноутбуки, мобильные устройства, настольные компьютеры и др. Каждое устройство имеет присвоенный ему уникальный идентификатор — MAC-адрес.

- В правой части рисунка вы видите централизованную систему для сбора и хранения данных от нескольких клиентов по кластерам Elasticsearch. Мы сфокусируемся на том, как правильно реализовать такой централизованный кластер и индекс, что позволит лучше понять их работу.
- Маршрутизаторы на каждой клиентской площадке собирают дополнительные показатели для каждого подключенного устройства: объем скачанных и переданных данных, а также ссылки URL или доменные имена, посещаемые клиентом в определенный интервал времени. Маршрутизаторы Wi-Fi собирают такие сведения и периодически отправляют их для длительного хранения и анализа на централизованный API-сервер.
- В момент передачи маршрутизаторами Wi-Fi эти данные состоят из нескольких полей: обычно это сведения о роутере и MAC-адреса конечных устройств, для которых собраны метрики. Сервер API рассматривает записи и добавляет в них больше полезной для аналитики информации и уже потом сохраняет в Elasticsearch. Например, поиск по MAC-адресу выполняется для того, чтобы найти ассоциированного пользователя, которому это устройство назначено в системе.

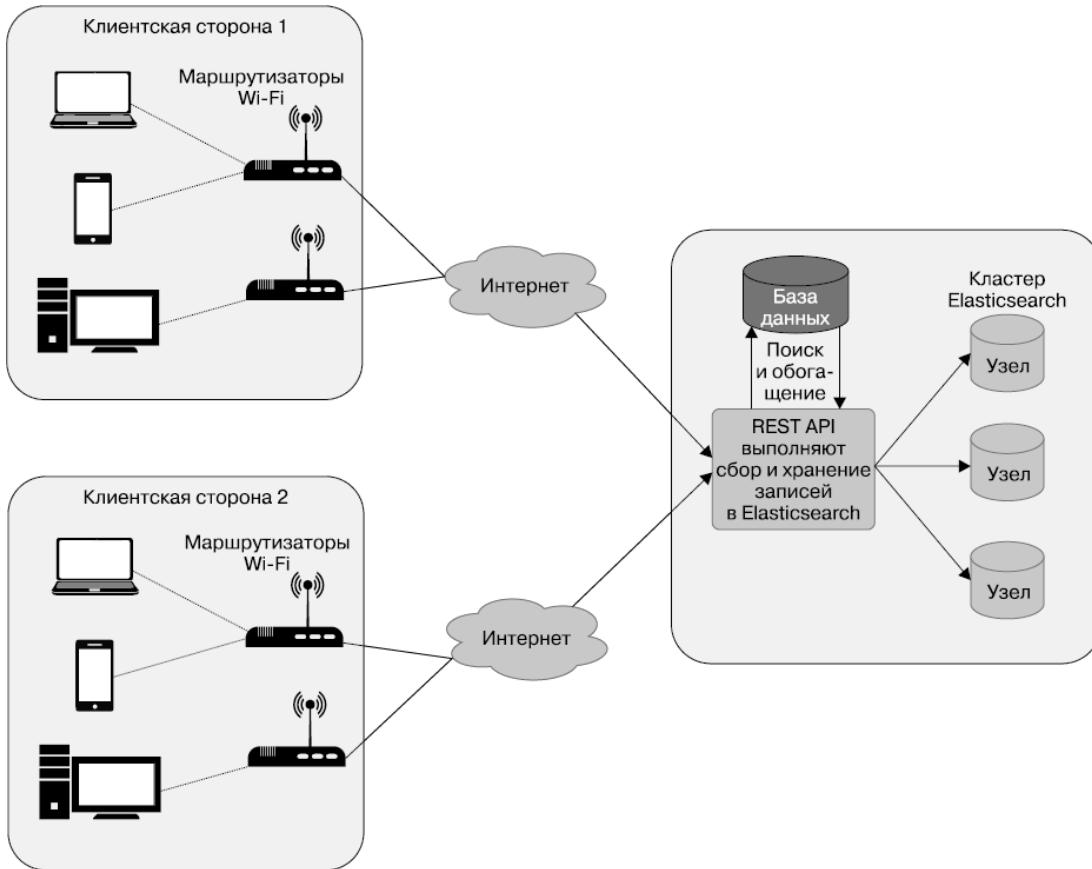


Рис. 4.1



Что такое метрика (показатель) и измерения? «Метрика» (metric) – это универсальный термин, который в мире аналитики используется для обозначения цифрового измерения. Простой пример метрики – объем трафика за определенный интервал времени. Термин «измерение» (dimension) обычно употребляется для определения дополнительной/внешней информации, зачастую – данных строкового типа. В текущем примере по MAC-адресу мы ищем дополнительную, связанную с ним информацию, а именно имя

пользователя, к которому устройство приписано в системе, а также название отдела, к которому принадлежит пользователь. Эти сведения и являются измерениями.

В результате подробные записи хранятся в Elasticsearch в виде структуры данных. Одна запись выглядит следующим образом:

```
"_source": {  
    "customer": "Google" // Клиент,  
    // которому принадлежит роутер Wi-Fi  
    "accessPointId": "AP-59484", //  
    // Идентификатор маршрутизатора  
    // или точки  
    // доступа Wi-Fi  
    "time": 1506148631061, // Время записи  
    // в миллисекундах со времени  
    // эпохи 1  
    // января 1970 года  
    "mac": "c6:ec:7d:c6:3d:8d", // MAC-адрес  
    // устройства клиента  
    "username": "Pedro Harrison", // Имя  
    // пользователя, которому назначено  
    // устройство  
    "department": "Operations", // Отдел  
    // пользователя,  
    // которому  
    // назначено устройство  
    "application": "CNBC", // Название  
    // приложения или домена  
    // для отчета  
    // трафика
```

```
        "category": "News", // Категория
приложения
        "networkId": "Internal", // SSID-
идентификатор сети
        "band": "5 GHz", // Диапазон 5
ГГц или 2,4 ГГц
        "location": "23.102789,72.595381", // Широта
и долгота,
        //
разделенные запятой
        "uploadTotal": 1340, // Байтов
отправлено с момента последнего отчета
        "downloadTotal": 2129, // Байтов принято
с момента последнего отчета
        "usage": 3469, // Общее
количество байтов, отправленных
        //
и принятых в
текущем периоде
        "uploadCurrent": 22.33, // Скорость
отправки в байтах/с в текущем периоде
        "downloadCurrent": 35.48, // Скорость приема
в байтах/с в текущем периоде
        "bandwidth": 57.82, // Общая скорость
в байтах/с
        //
(скорость
отправки + скорость приема)
        "signalStrength": -25, // Мощность
сигнала между маршрутизатором Wi-Fi
        //
и устройством
        ...
    }
```

Одна запись содержит различные метрики для выбранного устройства конечного пользователя и указанного промежутка времени.

Обратите внимание, что все данные в этом примере искусственно составленные. Несмотря на то что имена клиентов, пользователей и MAC-адресов выглядят реалистично, эти данные были созданы с использованием симулятора.

Теперь, когда вы знаете, что представляют собой приведенные данные и что значит каждая запись, приступим к загрузке данных в наш локальный экземпляр.

Загрузка данных с помощью Logstash

Для импортирования данных выполните шаги, перечисленные в сопроводительной информации к исходному коду этой книги (размещена в репозитории GitHub по следующей ссылке: <https://github.com/pranav-shukla/learningelasticsearch>).

Скачайте репозиторий с GitHub. Инструкции по импорту данных вы найдете в следующем каталоге: chapter-04/README.md.

После того как вы импортировали данные, необходимо верифицировать их с помощью следующего запроса:

```
GET /bigginsight/usageReport/_search
{
  "query": {
    "match_all": {}
  },
  "size": 1
}
```

Вы должны увидеть ответ следующего вида:

```
{  
  ...  
  "hits":  
  {  
    "total": 242835,  
    "max_score": 1,  
    "hits": [  
      {  
        "_index": "bigginsight",  
        "_type": "usageReport",  
        "_id": "AV7Sy4FofN33RK0L1VH0",  
        "_score": 1,  
        "_source": {  
          "inactiveMs": 1316,  
          "bandwidth": 51.03333333333333,  
          "signalStrength": -58,  
          "accessPointId": "AP-1D7F0",  
          "usage": 3062,  
          "downloadCurrent":  
            39.93333333333333,  
          "uploadCurrent": 11.1,  
          "mac": "d2:a1:74:28:c0:5a",  
          "tags": [],  
          "@timestamp": "2017-09-  
            30T12:38:25.867Z",  
          "application": "Dropbox",  
          "downloadTotal": 2396,  
          "@version": "1",  
          "networkId": "Guest",  
          "location": "23.102900,72.595611",  
          "time": 1506164775655,  
        }  
      }  
    ]  
  }  
}
```

```
        "band": "2.4 GHz",
        "department": "HR",
        "category": "File Sharing",
        "uploadTotal": 666,
        "username": "Cheryl Stokes",
        "customer": "Microsoft"
    }
}
]
}
```

Теперь, когда у вас есть необходимые данные, можете приступить к изучению различных типов агрегаций. Все запросы, использованные в этой главе, есть в материалах к этой книге, находятся в репозитории GitHub, в папке `chapter-04/queries.txt`. Эти запросы можно выполнить напрямую в Kibana Dev Tools, как вы уже видели ранее.

Метрические агрегации

Метрические агрегации работают с числовыми данными, вычисляя одну или несколько метрик в выбранном контексте. Контекст может быть запросом, фильтром или включать весь индекс/тип. Метрические агрегации можно вставлять в другие сегментарные агрегации. В таком случае выбранные метрики будут вычислены для каждого сегмента в пределах агрегации.

Мы начнем с простых метрических агрегаций без вложений. Когда далее в этой главе мы рассмотрим сегментарные агрегации, вы также узнаете, как вставлять в них метрические агрегации.

В текущем разделе мы изучим следующие типы метрических агрегаций:

- агрегации суммы, среднего, максимального и минимального значений;
- агрегации статистики, расширенной статистики;
- агрегация мощности.

Рассмотрим их по порядку.

Агрегации суммы, среднего, максимального и минимального значений

Поиск суммы поля, минимального или максимального значения или среднего числа являются довольно распространенными операциями. В SQL запрос для вычисления суммы выглядит следующим образом:

```
SELECT sum(downloadTotal) FROM usageReport;
```

Таким образом будет вычислена сумма поля `downloadTotal` по всем записям в таблице. Для этого необходимо пройтись по всем записям таблицы или по всем записям в выбранном контексте и добавить значения выбранных полей.

В Elasticsearch вы можете написать похожий запрос, используя агрегацию суммы.

Агрегация суммы

Вот как написать простую агрегацию суммы:

```
GET bigginsight/_search
```

```
{  
    "aggregations": {  
        "download_sum": {  
            "sum": {  
                "field": "downloadTotal"  
            }  
        }  
    },  
    "size": 0  
}
```

1. Элементы `aggs` или `aggregations` на верхнем уровне должны служить оберткой агрегации.
2. Дайте агрегации имя. В данном случае мы выполняем агрегацию суммы в поле `downloadTotal` и выбрали соответствующее имя `download_sum`. Вы можете назвать ее как угодно. Это поле пригодится, когда нам понадобится найти эту конкретную агрегацию в результатах ответа.
3. Мы делаем агрегацию суммы, следовательно, применяется элемент `sum`.
4. Мы хотим сделать агрегацию терминов по полю `downloadTotal`.
5. Укажите `size = 0`, чтобы предотвратить попадание в ответ необработанных результатов. Нам нужны только итоги агрегации, а не результаты поиска. Поскольку мы не указали никаких высокоранговых элементов `query`, запрос будет работать со всеми документами. Нам не нужны в ответе необработанные документы (или результаты поисковой выдачи).

Ответ должен выглядеть следующим образом:

```
{  
    "took": 92,  
    ...  
    "hits": {  
        "total": 242836, 1  
        "max_score": 0,  
        "hits": []  
    },  
    "aggregations": {2  
        "download_sum": {3  
            "value": 2197438700 4  
        }  
    }  
}
```

Разберемся в основных параметрах ответа.

1. Элемент `hits.total` показывает количество документов, соответствующих контексту запроса. Если не указан дополнительный запрос или фильтр, будут включены все документы в типе или индексе.
2. По аналогии с запросом этот ответ помещен внутрь агрегации для представления в таком виде.
3. Ответ запрошенной нами агрегации имеет название `download_sum`, следовательно, мы получаем наш ответ от агрегации суммы внутри элемента с таким же именем.
4. Фактическое значение выводится после применения агрегации суммы.

Агрегации среднего, максимального, минимального значений очень похожи. Кратко их рассмотрим.

Агрегация среднего значения

Агрегация среднего находит среднее значение по всем документам в контексте запроса:

```
GET bigginsight/_search
{
  "aggregations": {
    "download_average": {
      "avg": {
        "field": "downloadTotal"
      }
    },
    "size": 0
}
```

Заметные отличия от агрегации суммы состоят в следующем.

1. Мы выбрали другое название, `download_average`, чтобы было ясно, что данная агрегация призвана вычислить среднее значение.
2. Тип выполняемой агрегации — `avg` вместо `sum`, как в предыдущем примере.

Структура ответа идентична ответу из предыдущего подраздела, но в поле значения мы увидим среднее значение запрошенных полей.

Агрегации минимального и максимального значений аналогичны.

Агрегация минимального значения

Найдем минимальное значение поля `downloadTotal` во всем индексе/типе:

```
GET bigginsight/_search
{
  "aggregations": {
    "download_min": {
      "min": {
        "field": "downloadTotal"
      }
    }
  },
  "size": 0
}
```

Агрегация максимального значения

Найдем максимальное значение поля `downloadTotal` во всем индексе/типе:

```
GET bigginsight/_search
{
  "aggregations": {
    "download_max": {
      "max": {
        "field": "downloadTotal"
      }
    }
  },
  "size": 0
}
```

```
        }
    },
    "size": 0
}
```

Это очень простые агрегации. Теперь рассмотрим более усложненные агрегации статистики и расширенной статистики.

Агрегации статистики и расширенной статистики

Указанные агрегации вычисляют некоторые распространенные статистические значения в пределах одного запроса и без выполнения дополнительных запросов. Благодаря тому что статистические данные вычисляются в один заход, а не запрашиваются несколько раз, экономятся ресурсы Elasticsearch. Клиентский код также становится проще, если вы заинтересованы в нескольких видах таких данных. Взглянем на пример агрегации статистики.

Агрегация статистики

Агрегация статистики вычисляет сумму, среднее, максимальное, минимальное значение и общее количество документов в один заход:

```
GET bigginsight/_search
{
  "aggregations": {
    "download_stats": {
      "stats": {
        "field": "downloadTotal"
      }
    }
  }
}
```

```
},
"size": 0
}
```

Запрос статистики по структуре аналогичен другим метрическим агрегациям, о которых вы уже знаете; ничего особенного здесь не происходит.

Ответ должен выглядеть следующим образом:

```
{
  "took": 4,
  ...,
  "hits": {
    "total": 242836,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "download_stats
```

Как видите, ответ с элементом `download_stats` содержит общее количество, минимальное, максимальное, среднее значение и сумму. Такой вывод очень удобен, так как уменьшает количество запросов и упрощает клиентский код.

Взглянем на агрегацию расширенной статистики.

Агрегация расширенной статистики

Агрегация `extended stats` возвращает немного больше статистики в дополнение к предыдущему варианту:

```
GET bigginsight/_search
{
  "aggregations": {
    "download_estats": {
      "extended_stats": {
        "field": "downloadTotal"
      }
    }
  },
  "size": 0
}
```

Ответ будет выглядеть следующим образом:

```
{
  "took": 15,
  "timed_out": false,
  ...
  "hits": {
    "total": 242836,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "download_estats": {
```

```
        "count": 242835,
        "min": 0,
        "max": 241213,
        "avg": 9049.102065188297,
        "sum": 2197438700,
        "sum_of_squares": 133545882701698,
        "variance": 468058704.9782911,
        "std_deviation": 21634.664429528162,
        "std_deviation_bounds": {
            "upper": 52318.43092424462,
            "lower": -34220.22679386803
        }
    }
}
```

В ответе вы также получаете сумму квадратов, расхождение, стандартное отклонение и его границы.

Агрегация мощности

Подсчет уникальных элементов может быть выполнен с помощью агрегации мощности. Это похоже на поиск результата запроса, как показано ниже:

```
select count(*) from (select distinct username
from usageReport) u;
```

Определение мощности или количества уникальных значений для конкретного поля является довольно распространенной задачей. Например, если у вас есть поток кликов² (click-stream) от различных посетителей вашего сайта,

вы можете захотеть узнать, сколько уникальных посетителей на сайте в выбранный день, неделю или месяц.

Разберемся, как найти количество уникальных посетителей с помощью имеющихся данных сетевого трафика:

```
GET bigginsight/_search
{
  "aggregations": {
    "unique_visitors": {
      "cardinality": {
        "field": "username"
      }
    }
  },
  "size": 0
}
```

Ответ агрегации мощности выглядит так же, как и в других метрических агрегациях:

```
{
  "took": 110,
  ...
  "hits": {
    "total": 242836,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "unique_visitors": {
      "value": 79
    }
  }
}
```

```
}
```

```
}
```

Теперь, когда мы разобрались в простейших типах агрегаций, можем рассмотреть некоторые сегментарные агрегации.

Сегментарные агрегации

Сегментарные агрегации, или агрегации обобщения (bucket aggregations), полезны для анализирования того, как целое соотносится со своими частями, и получения более детальной картины. Они помогают разделять данные на небольшие части. Сегментарные агрегации всех типов разбивают данные на разные сегменты. Это наиболее распространенный вид агрегации, который используется в любом процессе анализа.

На базе примера данных сетевого трафика мы рассмотрим следующие темы.

- Сегментирование строковых данных.
- Сегментирование числовых данных.
- Агрегирование отфильтрованных данных.
- Вложенные агрегации.
- Сегментирование с нестандартными условиями.
- Сегментирование данных даты/времени.
- Сегментирование геопространственных данных.

Сегментирование строковых данных

Иногда требуется сегментировать данные или фрагмент данных, основанных на полях строкового типа, обычно это поля типа `keyword` в Elasticsearch. Ниже приведены некоторые примеры или сценарии применения, когда вам может понадобиться сегментировать данные по полю строкового типа.

- Сегментирование данных сетевого трафика по отделам.
- Сегментирование данных сетевого трафика по пользователям.
- Сегментирование данных сетевого трафика по приложениям или по категориям.

Самый распространенный способ сегментирования ваших данных строкового типа — с помощью агрегации терминов. Рассмотрим этот процесс детально.

Агрегация терминов (terms aggregation). Скорее всего, это самая популярная агрегация. Она полезна для сегментирования или группирования данных по выбранным определенным значениям. Предположим, в отношении примера с данными сетевого трафика, который мы загрузили, у нас есть следующий вопрос: *каковы наиболее распространенные категории сайтов, которые посетило большинство пользователей?*

Нам интересно узнать, какие сайты предпочитали пользователи, а не каков был объем трафика. В реляционной базе данных мы могли бы написать запрос следующего вида:

```
SELECT category, count(*) FROM usageReport  
GROUP BY category ORDER BY  
count(*) DESC;
```

В запросе агрегаций Elasticsearch для получения аналогичного результата запрос может быть записан так:

```
GET /bigginsight/usageReport/_search
{
  "aggs": {
    "byCategory": {
      "terms": {
        "field": "category"
      }
    }
  },
  "size": 0
}
```

1 2 3 4 5

Разберемся с условиями запроса.

1. Элементы `aggs` или `aggregations` на верхнем уровне должны обрамлять любую агрегацию.
2. Дадим агрегации имя; в данном случае мы выполняем агрегацию терминов по полю категории и, следовательно, выбрали имя `byCategory`.
3. Поскольку у нас агрегация терминов, применяется элемент `terms`.
4. Требуется сделать агрегацию терминов по полю `category`.
5. Укажем `size = 0`, чтобы предотвратить попадание в ответ необработанных результатов. Нам нужны только результаты агрегации, но не результаты поиска. Поскольку мы не указали никаких элементов `query` высшего уровня, запрос

будет работать со всеми документами. В результатах нам не нужны необработанные документы (или совпадения поиска).

Ответ должен выглядеть следующим образом:

```
{  
    "took": 11,  
    "timed_out": false,  
    "_shards": {  
        "total": 5,  
        "successful": 5,  
        "failed": 0  
    },  
    "hits": {  
        "total":  
        242835,  
        "max_score": 0,  
        "hits": [  
            {"  
                "id": 1,  
                "category": "Electronics",  
                "brand": "Dell",  
                "model": "XPS 13",  
                "year": 2018,  
                "price": 1200000,  
                "rating": 4.5,  
                "stock": true  
            },  
            {"  
                "id": 2,  
                "category": "Electronics",  
                "brand": "Dell",  
                "model": "XPS 13",  
                "year": 2018,  
                "price": 1200000,  
                "rating": 4.5,  
                "stock": true  
            },  
            {"  
                "id": 3,  
                "category": "Electronics",  
                "brand": "Dell",  
                "model": "XPS 13",  
                "year": 2018,  
                "price": 1200000,  
                "rating": 4.5,  
                "stock": true  
            },  
            {"  
                "id": 4,  
                "category": "Electronics",  
                "brand": "Dell",  
                "model": "XPS 13",  
                "year": 2018,  
                "price": 1200000,  
                "rating": 4.5,  
                "stock": true  
            },  
            {"  
                "id": 5,  
                "category": "Electronics",  
                "brand": "Dell",  
                "model": "XPS 13",  
                "year": 2018,  
                "price": 1200000,  
                "rating": 4.5,  
                "stock": true  
            }  
        ]  
    },  
    "aggregations": {  
        "byCategory": {  
            "buckets": [{"  
                "category": "Electronics",  
                "doc_count_error_upper_bound": 0,  
                "sum_other_doc_count": 0,  
                "buckets": [{"  
                    "brand": "Dell",  
                    "doc_count": 5  
                }]  
            }]  
        }  
    }  
}
```



```
        "doc_count": 19317
    }
]
}
}
}
```

Обратите внимание на следующие элементы ответа, отмеченные числами.

1. Элемент `total` под элементом `hits` (при навигации с вершины элемента JSON мы будем ссылаться на него как на `hits.total`) равен 242835. Это общее количество документов, обработанных агрегацией.
2. Массив `hits.array` пуст. Так произошло потому, что мы обозначили `"size": 0`, следовательно, сюда не были включены никакие результаты поиска. Мы заинтересованы в результатах агрегации, а не поиска.
3. Элемент `aggregations` на высшем уровне ответа JSON содержит все результаты агрегации.
4. Название агрегации — `byCategory`. Мы сами дали ей такое название. С его помощью будет легче ориентироваться в ответе запроса, поскольку ответ может быть создан для нескольких агрегаций одновременно.
5. Элемент `doc_count_error_upper_bound` отмечает ошибки во время агрегации. Данные разбиты на шарды; если каждый шард отправляет данные для всех ключей сегментирования, это приводит к слишком большому объему данных при отправке по сети. Elasticsearch отправляет только первые `n` сегментов по сети, если

агрегация была запрошена для первых n позиций. В данном случае n — это количество сегментов агрегации, определенное параметром `size`. Более детально мы рассмотрим этот параметр далее в главе.

6. `sum_other_doc_count` — общее количество документов, которые не попали в возвращенные сегменты. По умолчанию агрегации терминов возвращают первые десять сегментов, если доступно более десяти разных сегментов. Оставшиеся документы помимо этих десяти суммируются и возвращаются в это поле. В данном случае есть только восемь категорий и, следовательно, это поле равно 0.
7. Агрегация возвращает список сегментов.
8. Ключ одного из сегментов относится к категории `Chat`.
9. Элемент `doc_count` показывает количество документов в сегменте.

Как вы можете видеть, в результатах запроса есть только восемь разных сегментов.

Далее нам нужно узнать наиболее распространенные приложения с точки зрения максимального количества записей для каждого приложения:

```
GET /bigginsight/usageReport/_search?size=0
{
  "aggs": {
    "byApplication": {
      "terms": {
        "field": "application"
      }
    }
  }
}
```

```
        }
    }
}
```

Обратите внимание, что мы добавили `size=0` как параметр запроса прямо в URL.

Мы получаем следующий ответ:

```
{
  ...
  "aggregations": {
    "byApplication": {
      "doc_count_error_upper_bound": 6325,
      "sum_other_doc_count": 129002,
      "buckets": [
        {
          "key": "Skype",
          "doc_count": 26115
        },
        ...
      ]
    }
}
```

Обратите внимание, что `sum_other_doc_count` имеет большое значение — 129002. Это большое число по сравнению с суммарным количеством совпадений поиска. Как вы видели в предыдущем запросе, в индексе присутствует около 242 000 документов. Такая ситуация возникает по причине того, что по умолчанию эта агрегация терминов возвращает только десять сегментов. При текущих настройках первые десять сегментов с наиболее используемыми документами возвращаются в порядке убывания. Оставшиеся документы, не вошедшие в топ-10, обозначены в `sum_other_doc_count`. Фактически есть

30 разных приложений, которые используют наш сетевой трафик. Цифра в `sum_other_doc_count` — это количество оставшихся приложений, которые не попали в список сегментов.

Чтобы получить топ-*n* сегментов вместо десяти по умолчанию, мы можем использовать параметр `size` внутри агрегации терминов:

```
GET /bigginsight/usageReport/_search?size=0
{
  "aggs": {
    "byApplication": {
      "terms": {
        "field": "application",
        "size": 15
      }
    }
  }
}
```

Обратите внимание, что этот `size` (указанный внутри агрегации `terms`) отличается от `size`, указанного на верхнем уровне. На верхнем уровне параметр `size` используется для предотвращения попадания результатов поиска, а `size` внутри агрегации терминов указывает максимальное количество сегментов для возврата.

Агрегация терминов очень полезна для получения данных, которые можно использовать в графиках или диаграммах, если требуется анализировать относительные числа в полях строкового типа из определенного набора документов. В главе 7 вы узнаете, как применять агрегацию терминов для создания графиков и диаграмм.

Далее мы взглянем на то, как выполнять сегментирование полей числовых типов.

Сегментирование числовых данных

Еще один распространенный случай применения сегментированных агрегаций — необходимость разбиения данных на различные сегменты на основе числовых полей. Например, может потребоваться разбить данные о продуктах по различным ценовым диапазонам: до \$10, от \$10 до 50, от \$50 до 100 и т.д. Возможно также разделить данные по возрастным группам, количеству сотрудников и другим параметрам.

В этом разделе мы рассмотрим следующие агрегации:

- агрегацию гистограммы;
- агрегацию диапазона.

Агрегация гистограммы

Агрегация гистограммы может разбивать данные на разные сегменты на основании числового поля. В запросе вы можете настроить диапазон каждого фрагмента, который также может называться *интервалом*.

Итак, у нас есть сведения об объеме сетевого трафика. Поле `usage` хранит число байтов, которые были использованы для приема или передачи данных. Попробуем разделить эти данные на фрагменты:

```
POST /bigginsight/_search?size=0
{
  "aggs": {
```

```
"by_usage": {
    "histogram": {
        "field": "usage",
        "interval": 1000
    }
}
}
```

Агрегация выше будет разбивать все данные на следующие сегменты:

- от 0 до 999 — все записи, в которых `usage` ≥ 0 и < 1000 ;
- от 1000 до 1999 — все записи, в которых `usage` ≥ 1000 и < 2000 ;
- от 2000 до 3999 — все записи, в которых `usage` ≥ 2000 и < 3000 .

И т.д.

Ответ будет выглядеть следующим образом (сокращен):

```
{
    . . . ,
    "aggregations": {
        "by_usage": {
            "buckets": [
                {
                    "key": 0,
                    "doc_count": 30060
                },
                . . .
            ]
        }
    }
}
```

```
{  
    "key": 1000,  
    "doc_count": 42880  
},
```

Таким образом агрегация гистограммы создает сегменты равных значений, используя параметр `interval`, указанный в запросе. По умолчанию включаются все сегменты выбранного интервала, независимо от того, есть в них какие-либо документы или нет. Вы можете сделать так, чтобы получить только сегменты с документами. Для этого необходимо использовать параметр `min_doc_count`. Если он указан, агрегация гистограммы возвращает только те сегменты, в которых есть как минимум указанное количество документов.

Рассмотрим другой тип агрегаций — агрегацию диапазона, которую можно использовать для числовых данных.

Агрегация диапазона

Что, если мы не хотим, чтобы все сегменты имели один интервал? Вы можете создать неодинаковые по размеру сегменты с помощью агрегации диапазона.

Следующая агрегация разбивает данные на три сегмента: до 1 Кбайт, от 1 до 100 Кбайт и 100 Кбайт или больше. Обратите внимание, что в диапазоне мы можем указать `from` и `to`, но эти параметры необязательны. Если вы укажете только `to`, сегмент будет содержать все документы до указанного значения. Значение `to` эксклюзивно и не входит в текущий диапазон сегментов:

```
POST /bigginsight/_search?size=0  
{  
    "aggs": {
```

```
"by_usage": {
    "range": {
        "field": "usage",
        "ranges": [
            { "to": 1024 },
            { "from": 1024, "to": 102400 },
            { "from": 102400 }
        ]
    }
},
{
}
}
```

Ответ на данный запрос выглядит следующим образом:

```
{
    ...
    "aggregations": {
        "by_usage": {
            "buckets": [
                {
                    "key": "*-1024.0",
                    "to": 1024,
                    "doc_count": 31324
                },
                {
                    "key": "1024.0-102400.0",
                    "from": 1024,
                    "to": 102400,
                    "doc_count": 207498
                },
                ...
            ]
        }
    }
}
```

```
{  
    "key": "102400.0-*",  
    "from": 102400,  
    "doc_count": 4013  
}  
]  
}  
}  
}
```

Вы также можете создать свои метки key для сегментов диапазона, как в примере ниже:

```
POST /bigginsight/_search?size=0  
{  
    "aggs": {  
        "by_usage": {  
            "range": {  
                "field": "usage",  
                "ranges": [  
                    { "key": "Upto 1 kb", "to": 1024 },  
                    { "key": "1 kb to 100 kb", "from": 1024, "to": 102400 },  
                    { "key": "100 kb and more", "from": 102400 }  
                ]  
            }  
        }  
    }  
}
```

В результате полученные сегменты будут иметь набор ключей. Это удобно, если вы ищете релевантный сегмент в ответе и не хотите просматривать все возможные сегменты.

Для числовых данных доступно больше типов агрегаций, но рассмотрение всех не входит в задачи этой книги.

Далее мы разберемся в нескольких важных концепциях, связанных с сегментарной агрегацией и агрегациями в целом.

Агрегирование отфильтрованных данных

Немного отойдем от темы изучения различных сегментарных агрегаций и разберемся, как применять агрегации в отношении отфильтрованных данных. Вы уже научились применять агрегации для всех данных выбранного индекса/типа. На практике, прежде чем выполнять агрегации (метрические или сегментарные), вам почти всегда придется использовать некоторые фильтры.

Вернемся к примеру, который мы рассматривали в разделе «Сегментирование строковых данных». Мы нашли лидирующие категории по всему индексу и типу. Теперь нам необходимо найти такую категорию для конкретного клиента, а не для всех данных и всех клиентов:

```
GET /bigginsight/usageReport/_search?size=0
{
  "query": {
    "term": {
      "customer": "Linkedin"
    }
  },
  "aggs": {
    "byCategory": {
```

```
        "terms": {
            "field": "category"
        }
    }
}
```

Мы изменили исходный запрос, который искал лидирующие категории, дополнив его (выделен выше жирным шрифтом). Мы написали запрос, внутри которого добавлен фильтр по конкретному интересующему нас клиенту.

Когда мы применяем тип запроса с любым типом агрегаций, меняется контекст данных, над которыми выполняется агрегация. Решение, к каким данным будет применяться агрегация, принимается запросом/фильтром.

Чтобы лучше понять, как это работает, взглянем на ответ на этот запрос:

```
{
    "took": 18,
    ...
    "hits": {
        "total": 76607,
        "max_score": 0,
        "hits": []
    },
    ...
}
```

Общее количество результатов поиска в ответе сейчас намного ниже, чем в предыдущем запросе агрегации, проведенной по целому индексу/типу. Мы можем добавить

дополнительные фильтры к запросу для уменьшения затраченного времени.

Следующий запрос применяет несколько фильтров и сужает работу агрегации — в пределах клиента и конкретного подмножества временных интервалов:

```
GET /bigginsight/usageReport/_search?size=0
{
  "query": {
    "bool": {
      "must": [
        {"term": {"customer": "Linkedin"}},
        {"range": {"time": {"gte": 1506277800000, "lte": 1506294200000}}}
      ]
    }
  },
  "aggs": {
    "byCategory": {
      "terms": {
        "field": "category"
      }
    }
  }
}
```

Таким образом, с помощью фильтров вы можете настраивать охват агрегации. Далее мы продолжим изучать разные сегментарные агрегации и вы узнаете, как вставлять метрические агрегации в сегментарные.

Вложенные агрегации

Сегментарные агрегации разбивают контекст на два или более сегмента. Можно ограничить контекст агрегации, уточняя элемент запроса, как вы уже видели в предыдущем разделе.

Когда метрическая агрегация вставляется в сегментарную, метрическая составляющая считается в пределах каждого сегмента. Для улучшения понимания попробуем ответить на следующий вопрос. Каков общий объем израсходованного трафика для каждого пользователя или конкретного клиента в определенный день?

Нам нужно будет выполнить следующие действия.

1. Сначала отфильтровать все данные для выбранного клиента и дня. Для этого мы можем выполнить глобальный запрос булева типа.
2. После фильтрации данных необходимо создать сегменты по каждому пользователю.
3. Как только это сделано, мы можем вычислить метрическую агрегацию суммы для поля общего потребления трафика (которое включает прием и передачу данных).

Следующий запрос выполняет все перечисленное. Вы можете свериться с цифрами и сносками по трем основным целям запроса:

```
GET /bigginsight/usageReport/_search?size=0
{
    "query": 1
    "bool": {
```

```

    "must": [
        {"term": {"customer": "Linkedin"}},
        {"range": {"time": {"gte": 1506257800000, "lte": 1506314200000}}}
    ]
}
},
"aggs": {
    "by_users": {
        "terms": {
            "field": "username"
        },
        "aggs": {
            "total_usage": {
                "sum": { "field": "usage" }
            }
        }
    }
}
}

```

Обратите внимание, что на верхнем уровне агрегация `by_users`, являющаяся агрегацией терминов, содержит еще один элемент `aggs` с метрической агрегацией `total_usage` внутри.

Ответ должен выглядеть следующим образом:

```
{
    ...
    "aggregations": {

```

```
"by_users": {
    "doc_count_error_upper_bound": 0,
    "sum_other_doc_count": 453,
    "buckets": [
        {
            "key": "Jay May",
            "doc_count": 2170,
            "total_usage": {
                "value": 6516943
            }
        },
        {
            "key": "Guadalupe Rice",
            "doc_count": 2157,
            "total_usage": {
                "value": 6492653
            }
        },
        ...
    }
}
```

Как видите, каждый сегмент агрегации терминов содержит дочерний элемент `total_usage`, хранящий значение метрической агрегации. Сегменты сортируются по количеству документов в каждом сегменте в порядке убывания. Вы можете изменить порядок сегментов, отредактировав соответствующий параметр в пределах сегментарной агрегации.

Взгляните на следующий фрагмент запроса, измененного для сортировки сегментов в порядке убывания метрики `total_usage`:

```
GET /bigginsight/usageReport/_search
```

```
{
  ...
  "aggs": {
    "by_users": {
      "terms": {
        "field": "username",
        "order": { "total_usage": "desc" }
      },
      "aggs": {
        ...
        ...
      }
    }
  }
}
```

Выделенный пункт `order` в порядке убывания сортирует сегменты, используя вставленную агрегацию `total_usage`.

Сегментарные агрегации могут быть вставлены в другие агрегации такого же типа. Для улучшения понимания попробуем ответить на следующий вопрос: какие два пользователя в каждом отделе потребляют больше всего трафика? Следующий запрос поможет нам получить ответ:

```
GET /bigginsight/usageReport/_search?size=0
{
  "query": {
    "bool": {
      "must": [
        {"term": {"customer": "Linkedin"}},
        {"range": {"time": {"gte": 1506257800000, "lte": 1506314200000}}}
      ]
    }
  }
}
```

```

        }
    },
    "aggs": {
        "by_departments": 2
    {
        "terms": { "field": "department" },
        "aggs": {
            "by_users": 3
{
            "terms": {
                "field": "username",
                "size": 2,
                "order": { "total_usage": "desc" }
            },
            "aggs": {
                "total_usage": { "sum": { "field": "usage" } } 4
            }
        }
    }
}
}
}

```

Обратите внимание на следующие пункты по числам в запросе.

1. Запрос, фильтрующий по конкретному клиенту и диапазону времени.
2. Агрегация терминов верхнего уровня для получения сегмента по каждому отделу.

3. Агрегация терминов второго уровня для получения топ-2 пользователей (обратите внимание на `size = 2`) в пределах каждого сегмента.
4. Метрическая агрегация, которая вычисляет общий объем трафика в пределах изначального сегмента. Для подсчета общего объема для каждого пользователя предназначена агрегация `by_users`, являющаяся текущим изначальным сегментом агрегации `total_usage`.

Таким образом мы можем вкладывать сегментарные и метрические агрегации друг в друга, чтобы быстро и эффективно получать ответы на сложные вопросы касательно хранимых в Elasticsearch больших данных.

Сегментирование с нестандартными условиями

Иногда нам нужно еще больше контроля над тем, как создаются сегменты. Уже рассмотренные нами агрегации работали с одним типом или полем. Если поле, на основе которого мы хотим разбить данные, строкового типа, мы обычно используем агрегацию терминов. Если поле числового типа, то есть несколько возможных вариантов, таких как агрегация гистограммы, агрегация диапазона и др., позволяющие разбивать данные на различные сегменты.

Следующие агрегации позволяют создать один или несколько сегментов на основании выбранных нами запросов/фильтров.

- Агрегация фильтра.
- Агрегация фильтров.

Рассмотрим их по очереди.

Агрегация фильтра

Зачем использовать агрегацию фильтра? Такой тип агрегации позволяет нам создать один сегмент по произвольному фильтру и оценить конкретные метрики в пределах этого сегмента.

Например, если бы мы хотели создать сегмент всех записей для категории Chat, мы могли бы использовать фильтр терминов. Мы хотим создать сегмент со всеми записями, которые соответствуют условию category = Chat.

```
POST /bigginsight/_search?size=0
{
  "aggs": {
    "chat": {
      "filter": {
        "term": {
          "category": "Chat"
        }
      }
    }
  }
}
```

Ответ должен выглядеть следующим образом:

```
{
  "took": 4,
  ...
  "hits": {
```

```
    "total": 242836,
    "max_score": 0,
    "hits": []
},
"aggregations": {
    "chat": {
        "doc_count": 52277
    }
}
```

Как видите, элемент `aggregations` содержит только один пункт, относящийся к категории `Chat`. Он включает 52 277 документов. Этот ответ можно увидеть как подмножество ответа агрегации терминов, которое содержит все категории, кроме `Chat`.

Рассмотрим агрегацию фильтров, которая позволяет сегментировать данные по нескольким фильтрам сразу.

Агрегация фильтров

С агрегацией фильтров вы можете создавать несколько сегментов, и каждый из них будет иметь свой фильтр, который сможет направлять документы в определенные для них сегменты. Рассмотрим пример.

Мы хотим создать несколько сегментов, чтобы узнать, сколько сетевого трафика использовано в категории `Chat`. В то же время мы хотим узнать, какая доля приходится на приложение `Skype` по сравнению с остальными приложениями в категории `Chat`. Для этой цели мы можем использовать агрегацию фильтров, поскольку она позволяет нам писать произвольные фильтры для создания сегментов:

```

GET bigginsight/_search?size=0
{
  "aggs": {
    "messages": {
      "filters": {
        "filters": {
          "chat": { "match": { "category": "Chat" } },
          "skype": { "match": { "application": "Skype" } },
          "other_than_skype": {
            "bool": {
              "must": { "match": { "category": "Chat" } },
              "must_not": { "match": { "application": "Skype" } }
            }
          }
        }
      }
    }
  }
}

```

Мы создали три фильтра для трех необходимых нам сегментов.

- *Сегмент с ключом chat.* Мы указали для фильтра category = Chat. Использованный нами запрос соответствия является высокоуровневым запросом, который понимает разметку базового поля. В данном случае категория базового поля —

`keyword` и, следовательно, запрос соответствия ищет конкретное определение `Chat`.

- *Сегмент с ключом Skype.* Мы указали для фильтра `application = Skype` и включили только трафик `Skype`.
- *Сегмент с ключом other_than_skype.* Здесь мы использовали булев запрос для фильтрации документов в категории `Chat`, но не `Skype`.

Как вы можете видеть, агрегация фильтров является мощным инструментом, если требуется получить нестандартные сегменты с применением разных фильтров. Таким образом вы получаете полный контроль над процессом сегментирования. Вы можете выбрать свои поля и свои условия для создания сегментов по индивидуальным требованиям.

Далее мы разберемся, как разбивать данные в столбцах типа даты по различным временным интервалам.

Сегментирование данных даты/времени

Вы уже видели, как сегментировать (или разбивать) ваши данные на разные типы столбцов/полей. Анализ данных с учетом времени является еще одной распространенной задачей. В процессе работы могут возникать вопросы, для ответа на которые необходимо применять агрегации с учетом даты и времени. Вопросы могут быть следующими.

- Как меняются объемы продаж в разное время?
- Как меняется прибыль от месяца к месяцу?

В контексте примера с сетевым трафиком с помощью анализа данных вы можете получить ответы на следующие вопросы.

- Как в зависимости от времени меняются требования к пропускной способности сети в моей организации?
- Какие приложения в определенный период времени использовали больше всего трафика?

В Elasticsearch доступна мощная агрегация даты/гистограммы (date histogram aggregation), которая может помочь найти ответы на такие вопросы. Давайте детально рассмотрим пример ее использования.

Агрегация даты/гистограммы

Используя агрегацию даты/гистограммы, мы узнаем, как создавать сегменты по полю с датой. В процессе мы также рассмотрим такие задачи, как:

- создание сегментов по периодам времени;
- использование разных часовых поясов;
- вычисление других показателей в пределах «нарезанных» временных интервалов;
- фокусировка на определенном дне и изменение интервалов.

Создание сегментов по периодам времени

Следующий запрос разобьет данные по интервалам в один день. По тому же принципу, как мы уже выполняли

сегментирование разных строковых значений, следующий запрос создаст сегменты по различным значениям времени, группируя их по интервалам в один день:

```
GET          /bigginsight/usageReport/_search?  
size=0      1  
{  
  "aggs": {  
    "counts_over_time": {  
      "date_histogram":  
        {  
          "field": "time",  
          "interval":  
            "1d"           3  
          }  
        }  
      }  
  }  
}
```

1. Мы указали `size = 0` как параметр запроса, вместо того чтобы задавать его в теле запроса.
2. Мы используем агрегацию `date_histogram`.
3. Мы хотим разбить данные по дням, поэтому задаем интервал как `1d` (1 день). Можно указывать интервалы следующим образом: `1d` (1 день), `1h` (1 ч), `4h` (4 ч), `30m` (30 мин) и т.д., то есть выбор динамических критериев достаточно широк.

Ответ на данный запрос должен выглядеть следующим образом:

```
{
  ...
  "aggregations": {
    "counts_over_time": {
      "buckets": [
        {
          "key_as_string": "2017-09-23T00:00:00.000Z",
          "key": 1506124800000,
          "doc_count": 62493
        },
        {
          "key_as_string": "2017-09-24T00:00:00.000Z",
          "key": 1506211200000,
          "doc_count": 5312
        },
        {
          "key_as_string": "2017-09-25T00:00:00.000Z",
          "key": 1506297600000,
          "doc_count": 175030
        }
      ]
    }
  }
}
```

Как видите, имеющиеся в нашем распоряжении данные охватывают лишь период в три дня. Сегменты в ответе содержат ключи двух видов: `key` и `key_as_string`. Поле `key`

для миллисекунд с начала времени эпохи UNIX (1 января 1970 года), а `key_as_string` — для начала отсчета временного интервала по Всемирному координированному времени (UTC). В нашем случае был выбран интервал один день. Первый сегмент с ключом `2017-09-23T00:00:00.000Z` является сегментом, содержащим документы между 23 и 24 сентября 2017 года.

Использование разных часовых поясов

Предположим, нужно разбить данные, ориентируясь на часовой пояс Индии (IST), а не на UTC. Для этого необходимо изменить параметр `time_zone`. Следует выбрать необходимое смещение указанного часового пояса относительно UTC. В данном случае оно составит `+05:30`, поскольку время IST отличается от UTC на 5 ч 30 мин:

```
GET /bigginsight/usageReport/_search?size=0
{
  "aggs": {
    "counts_over_time": {
      "date_histogram": {
        "field": "time",
        "interval": "1d",
        "time_zone": "+05:30"
      }
    }
  }
}
```

Теперь ответ выглядит следующим образом:

```
{
```

```

    . . . ,
  "aggregations": {
    "counts_over_time": {
      "buckets": [
        {
          "key_as_string": "2017-09-23T00:00:00.000+05:30",
          "key": 1506105000000,
          "doc_count": 62493
        },
        {
          "key_as_string": "2017-09-24T00:00:00.000+05:30",
          "key": 1506191400000,
          "doc_count": 0
        },
        {
          "key_as_string": "2017-09-25T00:00:00.000+05:30",
          "key": 1506277800000,
          "doc_count": 180342
        }
      ]
    }
  }
}

```

Как видите, ключи `key` и `key_as_string` для всех сегментов изменились. Теперь они указывают на начало дня по времени IST. Кроме того, отсутствуют документы за 24 сентября 2017 года, так как это воскресенье.

Подсчет других показателей в пределах выбранных интервалов времени

Пока мы разбили данные на фрагменты, используя лишь агрегацию даты/гистограммы для создания сегментов по полю времени. Таким образом, мы получили количество документов в каждом сегменте. Далее попробуем ответить на вопрос: каков общий объем трафика за день для выбранного клиента? Следующий запрос позволит нам получить ответ:

```
GET /bigginsight/usageReport/_search?size=0
{
  "query": { "term": { "customer": "Linkedin" } },
  "aggs": {
    "counts_over_time": {
      "date_histogram": {
        "field": "time",
        "interval": "1d",
        "time_zone": "+05:30"
      },
      "aggs": {
        "total_bandwidth": {
          "sum": { "field": "usage" }
        }
      }
    }
  }
}
```

Мы добавили фильтр термина для выбора данных только по одному клиенту. В пределах агрегации `date_histogram` мы

разместили другую метрическую агрегацию — агрегацию суммы — для подсчета общего объема расходуемого трафика в пределах каждого сегмента. Таким образом мы получим общий объем данных за каждый день. Далее вы можете видеть сокращенный ответ на запрос:

```
{  
    ...  
    "aggregations": {  
        "counts_over_time": {  
            "buckets": [  
                {  
                    "key_as_string": "2017-09-  
23T00:00:00.000+05:30",  
                    "key": 1506105000000,  
                    "doc_count": 18892,  
                    "total_bandwidth": {  
                        "value": 265574303  
                    }  
                },  
                ...  
            ]  
        }  
    }  
}
```

Фокусировка на определенном дне и изменение интервалов

Далее мы узнаем, как фокусироваться на определенном дне, фильтруя данные для других периодов времени и уменьшая значения интервала. Попытаемся получить почасовое

распределение по объему передачи данных 25 сентября 2017 года.

То, что мы сейчас делаем, называется *детализацией данных* (*drilling down*). В частности, результат предыдущего запроса можно представить в виде линейного графика, где время будет на оси X, а трафик — на оси Y. Если мы хотим рассмотреть конкретный день на графике, пригодится следующий запрос:

```
GET /bigginsight/usageReport/_search?size=0
{
  "query": {
    "bool": {
      "must": [
        {"term": {"customer": "Linkedin"}},
        {"range": {"time": {"gte": 1506277800000}}}
      ]
    }
  },
  "aggs": {
    "counts_over_time": {
      "date_histogram": {
        "field": "time",
        "interval": "1h",
        "time_zone": "+05:30"
      },
      "aggs": {
        "hourly_usage": {
          "sum": { "field": "usage" }
        }
      }
    }
  }
}
```

```
        }
    }
}
```

Сокращенный ответ будет выглядеть следующим образом:

```
{
    ...
    "aggregations": {
        "counts_over_time": {
            "buckets": [
                {
                    "key_as_string": "2017-09-25T00:00:00.000+05:30",
                    "key": 1506277800000,
                    "doc_count": 465,
                    "hourly_usage": {
                        "value": 1385524
                    }
                },
                {
                    "key_as_string": "2017-09-25T01:00:00.000+05:30",
                    "key": 1506281400000,
                    "doc_count": 478,
                    "hourly_usage": {
                        "value": 1432123
                    }
                },
                ...
            }
        }
    }
}
```

Как видите, теперь у нас есть сегменты с данными с почасовым интервалом.

С помощью агрегации даты/гистограммы вы можете выполнять различные виды анализа данных, ориентируясь на время или дату. Как вы увидели в примерах, задать агрегацию с интервалом от одного часа до одного дня довольно просто. Вы можете разбивать данные по конкретным интервалам по необходимости, без предварительного планирования. Есть возможность делать это и с большими данными — вряд ли существуют другие системы хранения данных, предлагающие такой уровень гибкости при работе с большими данными.

Сегментирование геопространственных данных

Еще одной полезной функцией является возможность геопространственного анализа данных. Если у вас есть данные типа геоточки с указанными координатами, вы можете выполнять различные виды их анализа с возможностью нанесения на карту для улучшения понимания данных.

Рассмотрим два типа геопространственных агрегаций:

- по геодистанции;
- по сетке GeoHash.

Агрегация по геодистанции

Агрегация по геодистанции позволяет создавать сегменты по расстоянию от выбранной геоточки. Для лучшего понимания взгляните на рис. 4.2, где показана агрегация по геодистанции с указанными параметрами `to` (слева), а также `to` и `from` (справа).

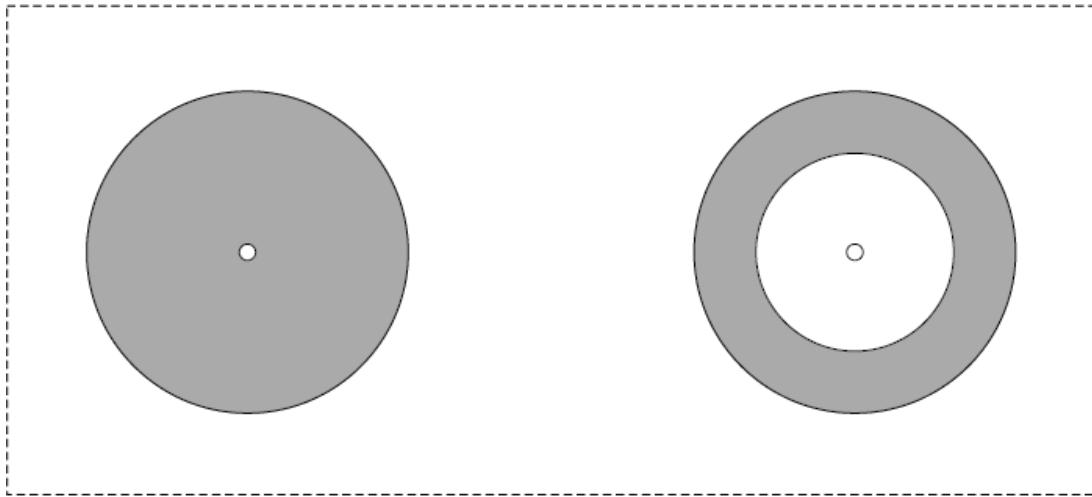


Рис. 4.2

Зона, обозначенная серым цветом, представляет собой область, включенную в агрегацию по геодистанции.

Следующая агрегация сформирует сегмент со всеми документами в пределах выбранного расстояния от конкретной геоточки. Результат совпадет с первым кругом (*слева*) на рис. 4.2. Закрашенная область от центра до края (выбранного радиуса) формирует круг:

```
GET bigginsight/usageReport/_search?size=0
{
  "aggs": {
    "within_radius": {
      "geo_distance": {
        "field": "location",
        "origin": {"lat": 23.102869, "lon": 72.595692},
        "ranges": [{"to": 5}]
      }
    }
  }
}
```

Как видите, параметр `ranges` работает как агрегация диапазона, которую мы рассматривали ранее. Он включает все точки в радиусе 5 м от выбранного `origin`. Это полезно в случае с агрегациями для подсчета всех возможных объектов в радиусе 2 км от выбранной точки, что часто используется на многих сайтах. Это хороший способ найти все организации в пределах выбранного расстояния от вашего местонахождения (например, все кофейни и больницы в радиусе 2 км).

Единица измерения дистанции по умолчанию — метры, но вы можете указать иное в параметре `unit`, например километры, мили или другие варианты.

Теперь взглянем, что получится, если мы укажем оба параметра, `from` и `to`, в агрегации геодистанции. Результат будет соответствовать кругу на рис. 4.2, *справа*:

```
GET bigginsight/usageReport/_search?size=0
{
  "aggs": {
    "within_radius": {
      "geo_distance": {
        "field": "location",
        "origin": {"lat": 23.102869, "lon": 72.595692},
        "ranges": [{"from": 5, "to": 10}]
      }
    }
  }
}
```

В данном случае мы сегментируем все точки, которые находятся не ближе чем 5 м и не далее чем 10 м от выбранной позиции. Аналогичным образом вы можете сформировать

сегмент точек, которые расположены на расстоянии как минимум x единиц измерения от выбранной позиции, указав только параметр `from`.

Рассмотрим агрегацию по сетке GeoHash.

Агрегация по сетке GeoHash

Агрегация по сетке GeoHash задействует механизм GeoHash для разбиения карты на небольшие блоки. Вы можете более детально узнать о системе GeoHash по ссылке <https://en.wikipedia.org/wiki/Geohash>. Эта система представляет карту мира в виде сетки прямоугольных областей с различной степенью точности. Низкие значения точности используются для крупных географических зон, а высокие значения — для мелких, более конкретизированных областей:

```
GET bigginsight/usageReport/_search?size=0
{
  "aggs": {
    "geo_hash": {
      "geohash_grid": {
        "field": "location",
        "precision": 7
      }
    }
  }
}
```

Мы установили уровень точности равным 7, поскольку данные в нашем примере с сетевым трафиком относятся к совсем небольшой географической области. Поддерживаются значения точности от 1 до 12. Взглянем на ответ на этот запрос:

```
{  
    ...,  
    "aggregations": {  
        "geo_hash": {  
            "buckets": [  
                {  
                    "key": "ts5e7vy",  
                    "doc_count": 161893  
                },  
                {  
                    "key": "ts5e7vw",  
                    "doc_count": 80942  
                }  
            ]  
        }  
    }  
}
```

После агрегации данных на блоки GeoHash с уровнем точности 7 все документы распределились по двум областям GeoHash, а также выполнен соответствующий подсчет документов, как вы видите в ответе. Мы можем уменьшить масштаб этой карты или запросить разбиение данных на меньшие фрагменты, увеличив значение точности.

Если вы попробуете использовать уровень точности 9, то увидите следующий ответ:

```
{  
    ...,  
    "aggregations": {  
        "geo_hash": {  
            "buckets": [  
                ...  
            ]  
        }  
    }  
}
```

```
{  
    "key": "ts5e7vy80k",  
    "doc_count": 131034  
},  
{  
    "key": "ts5e7vwrdb",  
    "doc_count": 60953  
},  
{  
    "key": "ts5e7vy84c",  
    "doc_count": 30859  
},  
{  
    "key": "ts5e7vwxfn",  
    "doc_count": 19989  
}  
]  
}  
}  
}
```

Как видите, агрегация по сетке GeoHash является довольно мощным инструментом агрегации данных по географическим регионам различных размеров и с разными уровнями точности. Вы также можете создать визуализацию с помощью Kibana или использовать эти данные в своем приложении с библиотекой, которая способна нанести данные на карту.

Итак, вы узнали о широком спектре сегментарных агрегаций, которые позволяют разбивать данные различных типов. Мы рассмотрели агрегации текстовых, числовых, геопространственных данных, а также данных типа «дата/

время». Далее вы узнаете, что представляют собой агрегации контейнеров.

Агрегации контейнеров

Агрегации контейнеров позволяют вам агрегировать результат других агрегаций. *Контейнером* называется результат одной агрегации при его использовании в качестве ввода для другой. Агрегация контейнеров является относительно новой функцией и до сих пор считается экспериментальной. Есть два типа агрегаций контейнеров.

- *Родительский* — агрегация контейнеров вложена в другую агрегацию.
- *Потомственный* — агрегация контейнеров является результатом изначальной агрегации.

Разберемся, как работает агрегация контейнеров, на примере агрегации общей суммы, которая является родительской.

Подсчет общей суммы использования трафика по времени. Когда мы изучали агрегацию даты/гистограммы в предыдущем разделе, мы рассмотрели агрегацию для подсчета почасового использования трафика в пределах одного дня. Выполнив задание, мы получили данные за 24 сентября с почасовым расходом трафика с 12 до 13, с 13 до 14 ч и т.д. Используя агрегацию общей суммы, мы также можем узнать общий расход трафика в конце каждого часа на протяжении дня. Взглянем на запрос и попробуем в нем разобраться:

```
GET /bigginsight/usageReport/_search?size=0
{
```

```
"query": {
    "bool": {
        "must": [
            {"term": {"customer": "Linkedin"}},
            {"range": {"time": {"gte": 1506277800000}}}
        ]
    }
},
"aggs": {
    "counts_over_time": {
        "date_histogram": {
            "field": "time",
            "interval": "1h",
            "time_zone": "+05:30"
        },
        "aggs": {
            "hourly_usage": {
                "sum": { "field": "usage" }
            },
            "cumulative_hourly_usage": {
                "1": {
                    "cumulative_sum": {
                        "2": {
                            "buckets_path": "3"
                        }
                    }
                }
            }
        }
    }
}
```

```
}
```

Только выделенный фрагмент кода является новым дополнением к запросу, который вы видели ранее. Нашей целью было подсчитать общую сумму по сегментам, созданным предыдущей агрегацией. Попробуем разобраться в новой части по указанным цифрам.

1. Даём понятное название для этой агрегации, размещаем ее внутри родительской агрегации даты/гистограммы.
2. Используем агрегацию общей суммы, следовательно, ссылаемся здесь на ее название, а именно — `cumulative_sum`.
3. Элемент `bucket_path` ссылается на метрику, основываясь на которой мы будем считать общую сумму. Нам необходимо суммировать полученные ранее значения метрики `hourly_usage`.

Ответ должен выглядеть следующим образом (сокращено):

```
{
  ...
  "aggregations": {
    "counts_over_time": {
      "buckets": [
        {
          "key_as_string": "2017-09-25T00:00:00.000+05:30",
          "key": 1506277800000,
          "doc_count": 465,
          "hourly_usage": {
```

```
        "value": 1385524
    },
    "cumulative_hourly_usage": {
        "value": 1385524
    }
},
{
    "key_as_string": "2017-09-
25T01:00:00.000+05:30",
    "key": 1506281400000,
    "doc_count": 478,
    "hourly_usage": {
        "value": 1432123
    },
    "cumulative_hourly_usage":
{
    "value": 2817647
}
}
```

Как видите, `cumulative_hourly_usage` содержит сумму `hourly_usage` до текущего момента. В первом сегменте почасовой и общий почасовой расходы равны. Во втором и последующих сегментах в общую сумму входят все почасовые сегменты.

Агрегации контейнеров являются мощным инструментом. Они могут высчитывать производные, скользящее среднее, среднее число по другим сегментам (минимальное, максимальное и пр.), а также среднее по ранее вычисленным агрегациям.

Резюме

Из этой главы вы узнали, как использовать Elasticsearch для создания мощных аналитических приложений. Вы увидели, как разбивать данные для их лучшего восприятия. Мы начали с метрической агрегации, чтобы разобраться с числовыми типами данных. Далее вы узнали о сегментарных агрегациях, о том, как данные «нарезаются» на фрагменты для упрощения поиска.

Мы также поговорили о работе агрегаций контейнеров. Все задачи мы выполняли, используя реалистичный набор данных о сетевом трафике. Благодаря приведенным примерам вы могли увидеть, насколько гибким является Elasticsearch как движок аналитики. Он позволяет анализировать любое поле без дополнительного моделирования данных, даже если речь идет о *больших данных*. Это исключительная возможность, которую могут предложить не все хранилища данных. Как вы увидите в главе 7, многие из рассмотренных здесь агрегаций можно использовать в Kibana.

Итак, теперь у вас есть отличный базис для изучения остальных компонентов Elastic Stack. Начиная со следующей главы, мы сменим акцент и перейдем к изучению утилиты Logstash, которая занимается сбором данных для Elasticsearch из разнообразных источников.

[2](#) Последовательность гиперссылок, по которой следует один посетитель сайта или несколько, представленная в порядке просмотра.

5. Анализ журнальных данных

Логи содержат подробную информацию о состоянии и работе системы или приложения. Каждая система/приложение записывает логи при любом событии; частота, объем и формат информации в логах меняются от одной системы/приложения к другой. С таким объемом информации в нашем распоряжении сбор логов, извлечение важной информации из них и их анализ в реальном времени может оказаться трудной задачей.

В предыдущих главах вы уже узнали, как Elasticsearch с его широкими возможностями агрегации помогает анализировать огромные объемы данных в псевдореальном времени. Прежде чем приступить к анализу, нужно найти инструмент, который может содействовать процессу сбора логов или упрощать его, извлекать важную информацию из логов и передавать ее в Elasticsearch.

В этой главе мы познакомимся с Logstash, еще одним ключевым компонентом Elastic Stack, который в основном используется как движок *ETL (Extract, Transform and Load — «извлечение, трансформация и загрузка»)*. Мы также рассмотрим следующие темы.

- Вызовы при анализе логов.
- Реакция Logstash на вызовы.
- Высокоуровневая архитектура Logstash.
- Плагины Logstash.
- Узел поглощения (ingest node) — новая функция Elasticsearch 5.x. Это легковесное решение для предварительной

обработки и пополнения информацией документов внутри Elasticsearch.

Вызовы при анализе логов

Логи — это записи об инцидентах или наблюдениях. Логи создаются широким спектром ресурсов, таких как системы, приложения, устройства, люди и т.д. Лог обычно содержит два показателя: метку времени (время записи события) и информацию о событии:

$$\text{Log} = \text{Timestamp} + \text{Data}$$

Логи используются для следующих целей.

- *Устранение неполадок.* Когда сообщается о баге или проблеме, в первую очередь информацию об этом нужно искать в логе. Например, при отслеживании стека исключений в логах можно легко найти причину проблемы.
- *Понимание поведения системы/приложения.* Во время работы приложения/системы лог работает как черный ящик. Изучая логи, вы поймете, что происходит внутри системы/приложения. Например, в логах можно отслеживать время, использованное разными фрагментами кода внутри приложения, и таким образом выявить узкие места и оптимизировать код для лучшей производительности.
- *Аудит.* Многие организации должны придерживаться тех или иных процедур проверки на соответствие и вынуждены вести логи. Например, входы пользователей в систему или их транзакции обычно хранятся в логах определенный промежуток времени, чтобы можно было провести аудит

или анализ злоумышленной деятельности пользователей/хакеров.

- *Предиктивная аналитика.* С развитием машинного обучения, искусственного интеллекта и методов извлечения данных появился новый тренд — предиктивная аналитика. Это область углубленной аналитики; специализируется на прогнозировании неизвестных событий, которые могут случиться в будущем. По результатам данных истории пользователя или транзакций возможно создать шаблоны и использовать их для выявления как возможностей, так и рисков в будущем. Предиктивная аналитика также позволяет организациям на основе полученных данных предугадывать результаты и поведение пользователей. Среди примеров применения предиктивной аналитики — предложения посетителям сайтов купить конкретную вещь или посмотреть фильм, выявление мошенничества, оптимизация маркетинговых кампаний и т.д.

Исходя из предыдущих примеров использования логов, можно прийти к выводу, что логи удобно задействовать для интенсивной обработки информации и применять для решения разного рода задач. Однако у логов есть свои особенности. Вот некоторые из них.

- *Нестандартный/несовместимый формат.* Каждая система записывает логи в своем собственном формате, поэтому администратору или конечному пользователю важно уметь разбираться в форматах логов, созданных каждой системой/приложением. Поскольку форматы различаются, поиск по разным типам логов может быть затруднительным. На рис. 5.1 показан типичный пример логов сервера SQL, исключений/логов Elasticsearch и логов NGNIX.

- *Децентрализованные логи.* Поскольку логи записываются широким спектром различных ресурсов, таких как системы, приложения, устройства и т.д., зачастую они разбросаны по нескольким серверам. С развитием облачных и распределенных вычислений стало намного сложнее выполнять поиск по логам, поскольку стандартные инструменты наподобие SSH и grep нельзя масштабировать. Следовательно, есть необходимость в централизованном управлении логами для упрощения поиска нужной информации аналитиками/администраторами.

```

2011-05-20 20:06:06.46 Server This instance of SQL Server last reported using a process ID of 1760 at 5/20/2011 8:03:41 PM C
2011-05-20 20:06:06.46 Server Registry startup parameters:
  -d C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\master.mdf
  -e C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\Log\ERRORLOG
  -l C:\Program Files\Microsoft SQL Server\MSSQL10_50.MSSQLSERVER\MSSQL\DATA\mastlog.ldf
2011-05-20 20:06:06.66 Server SQL Server is starting at normal priority base (-7). This is an informational message only. No

[2011-05-20 19:26:03,550][NAMAP][indexstore
integrityWithCommit [no]]
java.nio.file.FileSystemException: Path\is\full\or\exists\ 3.2\read\recycle\root\server\indexes\nc_hr_50b_data_nc2exp\3\index\segments_lg
at sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:185)
at sun.nio.fs.UnixException.rethrowAsIOException(UnixException.java:197)
at sun.nio.fs.UnixFileChannelImpl.open(UnixFileChannelImpl.java:177)
at java.nio.channels.FileChannel.open(FileChannel.java:287)
at java.nio.channels.FileChannel.open(FileChannel.java:281)
at org.elasticsearch.lucene.store.NormedKey.openInput(NormedKey.java:81)
at org.elasticsearch.lucene.store.FileWithChildDirectory.openInput(FileWithChildDirectory.java:184)
at org.elasticsearch.lucene.store.FileWithChildDirectory.openInput(FileWithChildDirectory.java:180)
at org.elasticsearch.index.store.StoreMetadataSnapshot.checkIndexForSaveFile(Store.java:930)
at org.elasticsearch.index.store.StoreMetadataSnapshot.createIndexForSaveFile(Store.java:909)
at org.elasticsearch.index.store.StoreMetadataSnapshot.init(Store.java:744)
at org.elasticsearch.index.store.Store.getMetadata(Store.java:233)
at org.elasticsearch.index.store.Store.getMetadata(Store.java:192)
at org.elasticsearch.indices.store.TransportModelAllocateSharedCoreMetaData.listCoreData(TransportModelAllocateSharedCoreMetaData.java:161)
at org.elasticsearch.indices.store.TransportModelAllocateSharedCoreMetaData.nodeByCoreAllocation(TransportModelAllocateSharedCoreMetaData.java:142)
at org.elasticsearch.indices.store.TransportModelAllocateSharedCoreMetaData.execute(TransportModelAllocateSharedCoreMetaData.java:171)
at org.elasticsearch.action.support.NodeOperation.TransportModelAllocateAction.execute(TransportModelAllocateAction.java:32)
at org.elasticsearch.action.support.NodeOperation.TransportModelAllocateAction.execute(TransportModelAllocateAction.java:290)
at org.elasticsearch.action.support.TransportModelAllocateAction.execute(TransportModelAllocateAction.java:256)
at org.elasticsearch.transport.RequestHandlerRegistry.processMessageReceived(RequestHandlerRegistry.java:179)
at org.elasticsearch.transport.netty.MessageChannelHandler$RequestHandler.handle(MessageChannelHandler.java:303)
at org.elasticsearch.common.util.concurrent.AbstractRunnable.run(AbstractRunnable.java:27)

93.180.71.3 - - [17/May/2015:08:05:23 +0000] "GET /downloads/product_1 HTTP/1.1" 304 0 "-" "Debian
APT-HTTP/1.3 (0.8.16-exp12ubuntu10.21)"
80.91.33.133 - - [17/May/2015:08:05:24 +0000] "GET /downloads/product_1 HTTP/1.1" 304 0 "-" "Debian
APT-HTTP/1.3 (0.8.16-exp12ubuntu10.17)"
```

Рис. 5.1

- *Несовместимый формат времени.* Логи хранят метки даты/времени, каждая система/приложение записывает время в своем формате, делая затруднительным идентификацию точного времени события (некоторые форматы более удобны для машин, чем для людей). Взаимосвязанные события происходят одновременно в разных системах. Вот лишь несколько примеров форматов времени, которые можно встретить в логах:

```
Nov 14 22:20:10
[10/Oct/2000:13:55:36 -0700]
172720538
053005 05:45:21
1508832211657
```

- *Неструктурированные данные.* Лог-данные не структурированы, и из-за этого может быть затруднен их анализ. Данные необходимо приводить к правильному виду прежде, чем потребуется выполнять анализ или поиск. Многие инструменты для анализа зависят от того, структурированы ли данные полностью или частично.

В следующем разделе вы узнаете, как Logstash может помочь в преодолении упомянутых проблем и облегчить процесс анализа логов.

Logstash

Logstash — популярный движок сбора данных с открытым исходным кодом. Он также имеет возможности работы с контейнерами в реальном времени. Logstash позволяет легко строить контейнеры для сбора данных из широкого спектра источников и обрабатывать их, дополнять, унифицировать и отправлять на хранение в различные места. Logstash предоставляет набор плагинов, известных как фильтры ввода и плагины вывода. Они довольно просты в использовании и облегчают процесс унифицирования и нормализации больших объемов разнообразных данных. Logstash выполняет работу движка ETL (рис. 5.2).

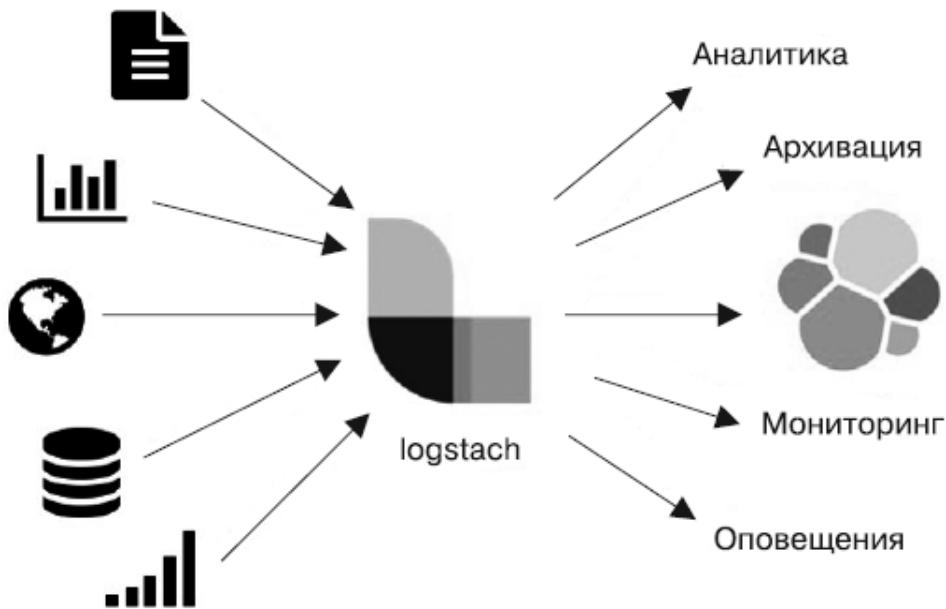


Рис. 5.2

Перечислим некоторые из особенностей Logstash.

- *Гибкая архитектура, основанная на плагинах.* Logstash содержит более 200 плагинов, разработанных компанией Elastic и сообществом открытого исходного кода. Вы можете использовать их для объединения, сопоставления и организации различных источников ввода, фильтров и выводов, а также создания контейнеров для обработки данных.
- *Расширяемость.* Компонент Logstash написан на JRuby и поддерживает гибкую архитектуру. Допускается создание собственных плагинов для персональных нужд.
- *Централизованная обработка данных.* Данные из любых источников можно легко извлекать с применением разнообразных плагинов ввода, а также дополнять их, изменять и отправлять различным получателям.

- **Универсальность.** Logstash поддерживает обработку всех типов логов, таких как логи Apache, NGINIX, системные логи, логи событий Windows. Он также предоставляет сбор метрик из широкого спектра приложений через TCP и UDP. Logstash может трансформировать запросы HTTP в события, поддерживает такие приложения, как Meetup, GitHub, JIRA и др. Возможен также сбор данных из имеющихся реляционных/NoSQL баз данных и очередей, включая Kafka, RabbitMQ и т.п. Контейнер обработки данных Logstash может быть легко масштабирован горизонтально. Начиная с версии Logstash 5, поддерживаются постоянные очереди, благодаря чему обеспечивается возможность надежно обрабатывать большие объемы входящих событий/данных.
- **Совместная работа.** Logstash имеет высокую связность с компонентами Elasticsearch, Beats и Kibana, что облегчает создание сквозных решений для анализа логов.

Установка и настройка

В следующих разделах мы рассмотрим установку и настройку Logstash в вашей системе.

Системные требования. Для работы Logstash необходимы Java runtime, Java 8. Убедитесь, что JAVA_HOME установлена как переменная среды. Для проверки своей версии Java выполните следующую команду:

```
java -version
```

Вы должны увидеть такой вывод:

```
java version "1.8.0_65"  
Java(TM) SE Runtime Environment (build  
1.8.0_65-b17)
```

Java HotSpot(TM) 64-Bit Server VM (build 25.65-b01, mixed mode)



Для Java вы можете использовать официальный дистрибутив Oracle (<http://www.oracle.com/technetwork/java/javase/downloads/index.html>) или дистрибутив с открытым исходным кодом, например OpenJDK (<http://openjdk.java.net/>).

Скачивание и установка Logstash

Скачать и установить Logstash не составит труда. Перейдите по ссылке <https://www.elastic.co/downloads/logstash#ga-release> и в зависимости от вашей операционной системы скачайте файл ZIP/TAR, как указано на рис. 5.3.

Download Logstash

Want to upgrade? We'll give you a hand. [Migration Guide »](#)

Version: 6.0.0

Release date: November 14, 2017

Notes: [View detailed release notes](#).
Not the version you're looking for? [View past releases](#).
Java 8 is required for Logstash 6.x and 5.x.

Downloads:

TAR.GZ sha	ZIP sha	DEB sha
RPM sha		

Рис. 5.3



Сообщество разработчиков Elastic динамично развивается, и релизы новых версий с обновленным функционалом и исправлениями выходят довольно часто. За время, которое вы уделили чтению этой книги, последняя версия Logstash могла измениться. Информация в книге актуальна для версии Logstash 6.0.0. Вы можете перейти по ссылке [past releases](#) (прошлые релизы) и скачать версию 6.0.0, если хотите продолжать как есть. Но материал хорошо подойдет и для любого релиза версии 6.x.

В отличие от Kibana, который требует совместимости с Elasticsearch как по основным, так и по младшим версиям, версии Logstash начиная с 5.6 совместимы с Elasticsearch. Полную схему совместимости вы можете найти по ссылке https://www.elastic.co/support/matrix#matrix_compatibility.

Установка под Windows

Распакуйте скачанный файл. После распаковки перейдите в созданный каталог, как показано в следующем фрагменте кода:

```
D:\>cd D:\packt\logstash-6.0.0
```



Мы будем указывать каталог установки Logstash как `LOGSTASH_HOME`.

Установка под Linux

Распакуйте пакет tar.gz и перейдите в созданный каталог, как показано ниже:

```
$> tar -xzf logstash-6.0.0.tar.gz  
$>cd logstash/
```

Запуск Logstash

Для запуска Logstash необходимо указать конфигурацию. Это можно сделать напрямую, используя параметр **-e** при указании конфигурационного файла (файл **.conf**) или указав параметр/флаг **-f**.

С помощью терминала/командной строки перейдите к **LOGSTASH_HOME/bin**. Убедитесь, что Logstash корректно работает после установки, выполнив следующую команду с простой конфигурацией (контейнер **logstash**) в качестве параметра:

```
D:\packt\logstash-6.0.0\bin>logstash -e 'input  
{ stdin { } } output {  
stdout {} }'
```

Вы должны получить следующие логи:

```
Sending Logstash's logs to D:/packt/logstash-  
6.0.0/logs which is now configured via  
log4j2.properties  
[2017-10-30T12:42:12,046][INFO] [logstash.modules.scaffold]
```

```
Initializing module {:module_name=>"fb_apache",
```

```
:directory=>"D:/packt/logstash-  
6.0.0/modules/fb_apache/configuration"}  
[2017-10-30T12:42:12,052][INFO ]  
[logstash.modules.scaffold]
```

```
Initializing module {:module_name=>"netflow",  
:directory=>"D:/packt/logstash-  
6.0.0/modules/netflow/configuration"}  
[2017-10-30T12:42:12,094][INFO ][logstash.agent]  
]
```

```
No persistent UUID file found. Generating new  
UUID  
{:uuid=>"fd6c25ed-6450-40fd-912a-c83bf2aec638",  
:path=>"D:/packt/logstash-6.0.0/data/uuid"}  
[2017-10-30T12:42:12,429][INFO ]  
[logstash.pipeline ]
```

```
Starting pipeline {"id"=>"main",  
"pipeline.workers"=>4,  
"pipeline.batch.size"=>125,  
"pipeline.batch.delay"=>5,  
"pipeline.max_inflight"=>500}  
[2017-10-30T12:42:12,490][INFO ]  
[logstash.pipeline ]
```

```
Pipeline main started  
The stdin plugin is now waiting for input:  
[2017-10-30T12:42:12,703][INFO ][logstash.agent]  
] Successfully started  
Logstash API endpoint {:port=>9600}
```

Теперь введите любой текст и нажмите клавишу **Enter**. Logstash добавляет к введенному текстовому сообщению метку времени и информацию об IP-адресе. Для выхода из Logstash нажмите **Ctrl+C** в оболочке, где была запущена утилита.

Итак, мы только что успешно запустили Logstash с парой простых конфигураций (контейнер). В следующем разделе мы подробнее поговорим о контейнерах в Logstash.

Архитектура Logstash

Контейнер обработки событий в Logstash имеет три стадии: ввода, фильтрации, вывода (рис. 5.4). Обязательны лишь ввод и вывод. Фильтрация является опциональной.

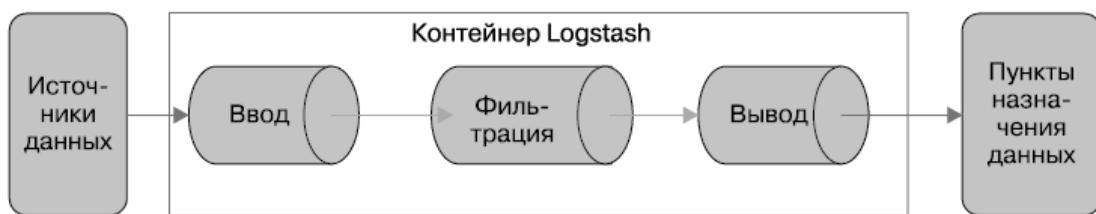


Рис. 5.4

Ввод создает события, *фильтры* модифицируют события ввода, *выводы* доставляют их в пункт назначения. Вводы и выводы поддерживают кодеки, которые позволяют кодировать или декодировать данные в момент их входа в контейнер или выхода из него без необходимости использования отдельного фильтра.

Для буферизации Logstash использует ограниченные очереди памяти между стадиями контейнеров по умолчанию (из ввода в фильтр и из фильтра в вывод). Если программа терминируется небезопасным способом, любые события в памяти будут потеряны. Для избежания потерь вы можете сохранять текущие

события на диск путем использования сохраняющихся очередей.



Сохраняющиеся очереди можно активизировать путем настройки `queue.type: persisted` в файле `logstash.yml`, который находится в папке `LOGSTASH_HOME/config`. `logstash.yml` – конфигурационный файл, который содержит настройки Logstash.

По умолчанию Logstash запускается с буфером размером 1 Гбайт. Вы можете изменить это в настройках `Xms` и `Xmx` в файле `jvm.options`, который находится в папке `LOGSTASH_HOME/config`.

Контейнер Logstash хранится в конфигурационном файле, имеющем расширение `.conf`. Ниже перечислены три секции конфигурационного файла:

```
input
{
}
filter
{
}
output
{}
```

Каждая из этих секций содержит одну или несколько конфигураций плагинов. Вы можете настроить плагин, указав его имя и далее настройки в виде пары «ключ/значение». Значение ключу присваивается с помощью оператора =>.

Возьмем тот же конфигурационный файл, который мы использовали в предыдущем разделе, и с некоторыми изменениями сохраним его в файл:

```
#simple.conf
#A simple logstash configuration

input {
    stdin { }
}

filter {
    mutate {
        uppercase => [ "message" ]
    }
}

output {
    stdout {
        codec => rubydebug
    }
}
```

Создадим папку conf в каталоге LOGSTASH_HOME. Создадим файл с названием simple.conf в папке LOGSTASH_HOME/conf.



На практике оптимально размещать все конфигурации в отдельных каталогах в папке LOGSTASH_HOME или в другом месте, вместо того чтобы хранить их в папке LOGSTASH_HOME/bin.

Возможно, вы обратили внимание, что файл содержит два обязательных элемента: ввод и вывод. В секции ввода фигурирует плагин с именем *stdin*, который принимает параметры по умолчанию. Секция вывода имеет плагин *stdout*, который принимает кодек *rubydebug*. Эти плагины используются для чтения из стандартного ввода и записи информации о событиях в стандартный вывод. Кодек *rubydebug* предназначен для вывода данных событий с помощью библиотеки Ruby Awesome Print. Он также содержит секцию фильтров с плагином *mutate*, который преобразует входящие сообщения событий в верхний регистр.

Запустим Logstash с новым контейнером/конфигурацией в файле `simple.conf`, как показано ниже:

```
D:\packt\logstash-6.0.0\bin>logstash -f  
./conf/simple.conf
```

Как только Logstash запущен, введите любой текст, например LOGSTASH IS AWESOME. Вы должны увидеть следующий ответ:

```
{  
    "@version" => "1",  
    "host" => "SHMN-IN",  
    "@timestamp" => 2017-11-03T11:42:56.221Z,  
    "message" => "LOGSTASH IS AWESOME\r"  
}
```

Как видите, вместе с сообщением ввода Logstash автоматически добавляет метку времени с указанием времени создания события, а также информацию о хосте и номере версии. Вывод имеет понятный вид благодаря кодеку *rubydebug*. Входящие события всегда сохраняются в поле под названием `message`.



Поскольку конфигурация была указана в файле, при запуске Logstash мы использовали флаг/параметр `-f`.

Обзор плагинов Logstash

Для Logstash существует обширная коллекция плагинов ввода, вывода, фильтрации, кодеков. Плагины доступны в виде автономных пакетов и размещаются на сайте **RubyGems.org**. По умолчанию многие распространенные плагины доступны как часть дистрибутива Logstash. Чтобы увидеть список плагинов, которые входят в текущую установку, выполните следующую команду:

```
D:\packt\logstash-6.0.0\bin>logstash-plugin list
```



Если к предыдущей команде добавить флаг `--verbose`, можно узнать версии каждого плагина.

Используя флаг `--group` с уточнением типа ввод, вывод, фильтр или кодек, мы можем узнать список установленных

плагинов соответствующего типа. Например:

```
D:\packt\logstash-6.0.0\bin>logstash-plugin  
list --group filter
```

Кроме того, можно найти все плагины по фрагменту названия, если указать его следующим образом в `logstash-plugin list`:

```
D:\packt\logstash-6.0.0\bin>logstash-plugin  
list 'pager'
```



В предыдущем примере команда `D:\packt\logstash-6.0.0\bin>` будет ссылаться на каталог `LOGSTASH_HOME\bin` на вашем устройстве.

Установка или обновление плагинов

Если вы не нашли необходимый плагин в конфигурации по умолчанию, можете установить его с помощью команды `bin\logstashplugin install`. Например, для установки плагина `logstash-output-email` выполните следующую команду:

```
D:\packt\logstash-6.0.0\bin>logstash-plugin  
install logstash-output-email
```

Используя команду `bin\logstash-plugin update` с названием плагина в качестве параметра, вы можете получить самую свежую версию плагина:

```
D:\packt\logstash-6.0.0\bin>logstash-plugin  
update logstash-output-s3
```



Если выполнить команду bin\logstash-plugin update без указания названия плагина, произойдет обновление всех плагинов.

Плагины ввода

Плагин ввода предназначен для настройки набора событий, которые передаются в Logstash. Этот плагин позволяет настроить один или несколько источников ввода. Он работает на месте первой секции, как это требуется в конфигурационном файле Logstash. В табл. 5.1 приведен список всех доступных при установке плагинов ввода.

Таблица 5.1

logstash-input-beats	logstash-input-couchdb_changes	logstash-input-elasticsearch	logstash-input-ganglia
logstash-input-xmpp	logstash-input-unix	logstash-input-syslog	logstash-input-stdin
logstash-input-udp	logstash-input-twitter	logstash-input-tcp	logstash-input-sqs
logstash-input-snmptrap	logstash-input-redis	logstash-input-pipe	logstash-input-log4j
logstash-input-s3	logstash-input-rabbitmq	logstash-input-lumberjack	logstash-input-http_poller
logstash-input-exec	logstash-input-file	logstash-input-http	logstash-input-imap
logstash-input-gelf	logstash-input-jdbc	logstash-input-irc	logstash-input-generator
logstash-input-heartbeat	logstash-input-graphite		

Более детальную информацию обо всех этих плагинах, а также о других доступных плагинах, которые не входят в стандартный дистрибутив, вы можете найти по следующей ссылке: <https://www.elastic.co/guide/en/logstash/6.0/input-plugins.html>.

Плагины вывода

Плагин вывода используется для отправки данных в требуемое место назначения. С его помощью можно настроить один или несколько источников вывода. Он работает на месте последней секции, как это предусмотрено в конфигурационном файле Logstash. В табл. 5.2 приведен список всех доступных при установке плагинов вывода.

Таблица 5.2

logstash-output-cloudwatch	logstash-output-nagios	logstash-output-irc	logstash-output-pagerduty
logstash-output-xmpp	logstash-output-tcp	logstash-output-stdout	logstash-output-redis
logstash-output-webhdfs	logstash-output-statsd	logstash-output-sns	logstash-output-rabbitmq
logstash-output-udp	logstash-output-sqs	logstash-output-s3	logstash-output-pipe
logstash-output-csv	logstash-output-graphite	logstash-output-file	logstash-output-elasticsearch
logstash-output-http			

Более детальную информацию обо всех этих плагинах, а также о других доступных плагинах, которые не входят в стандартный дистрибутив, вы можете найти по следующей ссылке: <https://www.elastic.co/guide/en/logstash/6.0/output-plugins.html>.

Плагины фильтров

Плагин фильтра используется для обработки и трансформации данных. Вы можете комбинировать один или несколько плагинов. Порядок размещения плагинов определяет порядок, в котором будут обрабатываться данные. Это промежуточная секция между вводом и выводом, она не является обязательной в конфигурации Logstash. В табл. 5.3 приведен список всех доступных при установке плагинов фильтрации.

Таблица 5.3

logstash-filter-cidr	logstash-filter-clone	logstash-filter-grok	logstash-filter-geoip
logstash-filter-date	logstash-filter-csv	logstash-filter-throttle	logstash-filter-xml
logstash-filter-fingerprint	logstash-filter-dns	logstash-filter-drop	logstash-filter-dissect
logstash-filter-syslog_pri	logstash-filter-useragent	logstash-filter-split	logstash-filter-translate
logstash-filter-uuid	logstash-filter-urldecode	logstash-filter-sleep	logstash-filter-ruby
logstash-filter-mutate	logstash-filter-metrics	logstash-filter-kv	logstash-filter-json

Более детальную информацию обо всех этих плагинах, а также о других доступных плагинах, которые не входят в стандартный дистрибутив, вы можете найти по следующей ссылке: <https://www.elastic.co/guide/en/logstash/6.0/filter-plugins.html>.

Плагины кодеков

Плагины кодеков используются для кодирования или декодирования входящих или исходящих событий в Logstash. Их также можно применять во вводе или выводе. Кодеки ввода предоставляют удобное декодирование ваших данных еще до

того, как они попадают на стадию ввода. Кодеки вывода обрабатывают данные прежде, чем они будут отправлены на стадию вывода. В табл. 5.4 приведен список всех доступных при установке плагинов кодеков.

Таблица 5.4

logstash-codec-netflow	logstash-codec-cef	logstash-codec-es_bulk	logstash-codec-dots
logstash-codec-collectd	logstash-codec-multiline	logstash-codec-msgpack	logstash-codec-line
logstash-codec-rubydebug	logstash-codec-json	logstash-codec-json_lines	logstash-codec-fluent
logstash-codec-plain	logstash-codec-graphite	logstash-codec-edn_lines	logstash-codec-edn

Более детальную информацию обо всех этих плагинах, а также о других доступных плагинах, которые не входят в стандартный дистрибутив, вы можете найти по следующей ссылке: <https://www.elastic.co/guide/en/logstash/6.0/codec-plugins.html>.

Обзор плагинов

В следующих разделах мы подробно разберем несколько наиболее распространенных плагинов ввода, вывода, фильтрации и кодеков.

Обзор плагинов ввода

Рассмотрим несколько часто используемых плагинов ввода.

File

Плагин File используется для трансляции событий из файлов строка за строкой. Он работает по принципу Linux/Unix-команд. Плагин отслеживает любые изменения в каждом файле и отправляет данные, начиная с того места, где файл был в последний раз прочитан. Он автоматически распознает сдвиг файлов. Доступна также опция чтения файла с самого начала.

Плагин File запоминает текущую позицию в каждом файле. Это происходит путем записи текущей позиции в отдельном файле с названием `sincedb`. Довольно удобно в том случае, если Logstash был перезапущен, продолжить работу с того места, на котором чтение файла было прекращено, при этом не упуская строки, добавленные в файл за то время, пока Logstash не работал.

Файл `sincedb` по умолчанию размещен в каталоге `<path.data>/plugins/inputs/file`, однако вы можете заменить этот путь любым другим, внеся изменения в параметр `sincedb_path`. Единственный обязательный параметр для этого плагина — `path`, указывающий на один или несколько файлов, из которыхчитываются данные.

Рассмотрим работу этого плагина на примере:

```
#sample configuration 1
#simple1.conf

input
{
  file{
    path => "/usr/local/logfiles/*"
  }
}
output
{
```

```
    stdout {  
        codec => rubydebug  
    }  
}
```

Предыдущая конфигурация дает указание транслировать все новые записи в файлы, находящиеся по адресу /usr/local/logfiles/:

```
#sample configuration 2  
#simple2.conf  
input  
{  
    file{  
        path => ["D:\es\app\*","D:\es\logs\*.txt"]  
        start_position => "beginning"  
        exclude => ["*.csv"]  
        discover_interval => "10s"  
        type => "applogs"  
    }  
}  
  
output  
{  
    stdout {  
        codec => rubydebug  
    }  
}
```

Эта конфигурация дает указание транслировать (streaming) все записи/строки логов в файлы, находящиеся по адресу D:\es\app*, а также только файлы типа .txt. Сюда же входят

файлы, найденные по адресу D:\es\logs*.txt. Записичитываются начиная с начала файла (указано параметром start_position => "beginning"), и в процессе поиска исключаются файлы типа .csv (указано параметром exclude => ["*.csv"], который получает массив значений). Каждая транслируемая строка будет сохранена в поле сообщений по умолчанию. Предыдущая конфигурация также дает указание добавлять один дополнительный тип поля со значением applogs (задано параметром type => "applogs"). Добавление дополнительного поля полезно при преобразовании событий в плагинах фильтрации или при идентификации событий для вывода. Параметр discover_interval используется для определения того, как часто будет расширяться путь для поиска новых файлов, созданных в каталоге, указанном под параметром path.



Если указать параметры start_position => "beginning" и since_db_path => "NULL", трансляция записей будет принудительно стартовать с начала файла при каждом перезапуске Logstash.

Beats

Плагин ввода Beats позволяет получать события из фреймворка Elastic Beats. Это коллекция легковесных демонов, используемых для сбора операционных данных с ваших серверов и доставки их в настраиваемые файлы вывода Logstash, Elasticsearch, Redis и др. Сюда входят такие компоненты, как Metricbeat, Filebeat, Winlogbeat и др. Filebeat

отправляет данные логов с ваших серверов. Metricbeat является агентом мониторинга серверов, он периодически собирает метрики по сервисам и операционным системам, запущенным на ваших серверах. Winlogbeat отправляет логи событий Windows. Более детально о фреймворке Beats и его компонентах мы поговорим в следующих главах.

Используя плагин ввода Beats, мы можем сделать так, чтобы Logstash прослушивал выбранные порты на предмет входящих соединений:

```
#beats.conf

input {
    beats {
        port => 1234
    }
}

output {
    elasticsearch {
    }
}
```

Единственным обязательным параметром для этого плагина является `port`. В приведенной конфигурации Logstash прослушивает входящие подключения `beats` и индексирует их в Elasticsearch. При запуске такой конфигурации вы можете заметить, что Logstash разворачивает входящий прослушиватель на порте 1234 в логах, как показано ниже:

```
D:\packt\logstash-6.0.0\bin>logstash -f
..../conf/beats.conf -r
```

Sending Logstash's logs to D:/packt/logstash-6.0.0/logs which is now configured via log4j2.properties

```
[2017-11-06T15:16:46,534][INFO      ] [logstash.modules.scaffold] Initializing module {:module_name=>"fb_apache", :directory=>"D:/packt/logstash-6.0.0/modules/fb_apache/configuration"}  
[2017-11-06T15:16:46,539][INFO      ] [logstash.modules.scaffold] Initializing module {:module_name=>"netflow", :directory=>"D:/packt/logstash-6.0.0/modules/netflow/configuration"}  
[2017-11-06T15:16:47,905][INFO      ] [logstash.pipeline ] Starting pipeline {"id"=>"main", "pipeline.workers"=>4, "pipeline.batch.size"=>125, "pipeline.batch.delay"=>5, "pipeline.max_inflight"=>500}  
[2017-11-06T15:16:48,491][INFO      ] [logstash.inputs.beats ] Beats inputs:  
Starting           input           listener  
{:address=>"0.0.0.0:1234"}  
[2017-11-06T15:16:48,554][INFO      ] [logstash.pipeline ] Pipeline main started  
[2017-11-06T15:16:48,563][INFO      ] [org.logstash.beats.Server] Starting server on port: 1234  
[2017-11-06T15:16:48,800][INFO      ][logstash.agent] Successfully started Logstash API endpoint {:port=>9600}
```

Logstash запускает входящий прослушиватель по адресу `0.0.0.0`, что является значением по умолчанию для параметра `host` этого плагина.

Вы можете запустить несколько прослушивателей, как видно ниже:

```
#beats.conf

input {
    beats {
        host => "192.168.10.229"
        port => 1234
    }
    beats {
        host => "192.168.10.229"
        port => 5065
    }
}

output {
    elasticsearch {
    }
}
```



Использование флага `-t` во время работы Logstash позволяет вам автоматически запускать конфигурацию, если она была изменена и сохранена. Это может быть полезно во время тестирования новых конфигураций без необходимости

вручную перезапускать Logstash каждый раз, когда конфигурация изменилась.

JDBC

Этот плагин предназначен для импорта данных из базы данных в Logstash. Каждая строка в результатах станет событием, а каждый столбец — полем внутри события. Используя этот плагин, можно импортировать все данные за один раз путем выполнения запроса или же настроить расписание для импорта с помощью синтаксиса cron (в параметре `schedule`). При задействовании этого плагина пользователю необходимо указать путь к драйверам JDBC, которые подходят для выбранной базы данных. Библиотека драйверов определяется с помощью параметра `jdbc_driver_library`.

Запрос SQL может быть указан с помощью параметра `statement` или сохранен в файл; путь к файлу задается в параметре `statement_filepath`. Для указания запроса можно использовать `statement` или `statement_filepath`. Хорошей практикой является сохранение больших запросов в файл. Этот плагин принимает только один запрос SQL, множественные запросы не поддерживаются. Если пользователю нужно выполнить несколько запросов для сбора данных из нескольких таблиц/видов, то необходимо указать несколько вводов JDBC (по одному на каждый запрос) в секции ввода конфигурации Logstash.

Размер результирующего набора указывается в параметре `jdbc_fetch_size`. Плагин будет «настраивать» на параметре `sql_last_value` для указания файла метаданных, который сохраняется в папке, заданной параметром `last_run_metadata_path`. После выполнения запроса этот файл будет обновлен с текущим значением `sql_last_value`.

Это значение используется для пошагового импорта данных из базы данных каждый раз, когда выполняется запрос, основанный на расписании schedule. Параметры запроса SQL могут быть указаны в настройке parameters, которая принимает хеш параметра запроса.

Взглянем на пример:

```
#jdbc.conf
input {
    jdbc {
        # path of the jdbc driver
        jdbc_driver_library => "/path/to/mysql-
connector-java-5.1.36-bin.jar"

        # The name of the driver class
        jdbc_driver_class      =>
"com.mysql.jdbc.Driver"

        # Mysql jdbc connection string to company
        database
        jdbc_connection_string      =>
"jdbc:mysql://localhost:3306/company"
        # user credentials to connect to the DB
        jdbc_user => "user"
        jdbc_password => "password"

        # when to periodically run statement, cron
        format (ex: every 30 minutes)
        schedule => "30 * * * *"

        # query parameters
        parameters => { "department" => "IT" }
```

```
# sql statement
statement => "SELECT * FROM employees WHERE
department= :department AND
created_at >= :sql_last_value"
}

}

output {
elasticsearch {
index => "company"
document_type => "employee"
hosts => "localhost:9200"
}
}
```

Приведенная конфигурация используется для подключения к схеме компании на базе модуля MySQLdb и предназначена для сбора записей работников из IT-отдела. Запрос SQL выполняется каждые 30 мин и проверяет, появились ли новые работники с момента последнего его запуска. Полученные строки отправляются в Elasticsearch и настраиваются как вывод.



По умолчанию до выполнения запроса параметр `sql_last_value` настраивается на четверг, 1 января 1970 года, и обновляется с меткой времени каждый раз, когда выполняется запрос. Вы можете принудительно задавать значение столбца, отличное от времени последнего выполнения. Для этого необходимо присвоить параметру `use_column_value` значение `true` и указать

название столбца, который будет использоваться параметром `tracking_column`.

IMAP

Данный плагин используется для чтения сообщений электронной почты с серверов IMAP. Он предназначен для получения писем, которые в зависимости от контекста сообщения или определенных отправителей могут условно обрабатываться Logstash и далее использоваться для поднятия тикетов JIRA, событий PagerDuty и т.д. Обязательные поля в конфигурации: `host`, `password` и `user`. В зависимости от требований сервера IMAP, к которому вы подключаетесь, возможно, понадобится указать значения дополнительных настроек, таких как `port`, `secure` и др. В параметре `host` вы указываете детали хоста IMAP, а `user` и `password`, соответственно, хранят пользовательские данные авторизации:

```
#email_log.conf
input {
    imap {
        host => "imap.packt.com"
        password => "secertpassword"
        user => "user1@pact.com"
        port => 993
        check_interval => 10
        folder => "Inbox"
    }
}

output {
    stdout {
```

```
    codec => rubydebug
}
elasticsearch {
    index => "emails"
    document_type => "email"
    hosts => "localhost:9200"
}
}
```

По умолчанию плагин `logstash-input-imap` считывает сообщения из папки `INBOX` и опрашивает сервер IMAP каждые 300 секунд. В приведенной выше конфигурации при указании параметра `check_interval` задается пользовательский интервал каждые 10 с. Каждое новое сообщение электронной почты воспринимается как событие и согласно конфигурации выше будет отправляться в стандартный вывод и в Elasticsearch.

Плагины вывода

В следующих подразделах мы подробно рассмотрим несколько наиболее распространенных плагинов вывода.

Elasticsearch

Этот плагин используется для передачи событий из Logstash в Elasticsearch. Это не единственный способ такой передачи, но самый предпочтительный. Как только данные поступят в Elasticsearch, их можно далее использовать для визуализации в Kibana. Плагин не имеет обязательных параметров, подключение к Elasticsearch происходит автоматически, применяется порт `localhost:9200`.

Простая конфигурация этого плагина выглядит следующим образом:

```
#elasticsearch1.conf
```

```
input {  
    stdin{  
    }  
}  
  
output {  
    elasticsearch {  
    }  
}
```

Нередко Elasticsearch размещается на других серверах, обычно безопасных, и может потребоваться хранить входящие данные в определенных индексах. Рассмотрим такой пример:

```
#elasticsearch2.conf
```

```
input {  
    stdin{  
    }  
}  
  
output {  
    elasticsearch {  
        index => "company"  
        document_type => "employee"  
        hosts => "198.162.43.30:9200"  
        user => "elastic"  
    }  
}
```

```
    password => "elasticpassword"
  }
}
```

Как видно в этом коде, входящие события будут сохраняться в индексе Elasticsearch с названием компании (задано с помощью параметра `index`) под типом `employee` (указано с помощью параметра `document_type`). Elasticsearch размещен по адресу `198.162.43.30:9200` (задано с помощью параметра `document_type`) и использует данные авторизации `elastic` и `elasticpassword` (указанны в параметрах `user` и `password`).

Если индекс по умолчанию не указан, то шаблон индекса будет `logstash-%(+YYYY.MM.dd)`, а `document_type` будет задан как тип события, даже если он уже существует, в ином случае тип документа будет назначен значением логов/событий.

Возможно также указать индекс `document_type` и `document_id` динамически, используя `syntax %({fieldname})`. В параметре `hosts` может быть указан список хостов. Протоколом по умолчанию будет HTTP, если иное четко не определено при настройке хостов.



Рекомендуется указывать узлы данных или узлы поглощения в поле `hosts`.

CSV

Этот плагин используется для хранения вывода в формате CSV. Обязательными его параметрами являются `path` (для указания папки сохранения файла вывода) и `fields` (задает названия

полей из событий, которые должны быть записаны в файл CSV). Если поле в событии не существует, будет записана пустая строка.

Рассмотрим пример. В следующей конфигурации Elasticsearch запрашивает по индексу "apachelogs" все документы, соответствующие statuscode:200. Поля "message", "@timestamp" и "host" записываются в файл .csv:

```
#csv.conf

input {
    elasticsearch {
        hosts => "localhost:9200"
        index => "apachelogs"
        query  => '{ "query": { "match": {
            "statuscode": 200 } }}'
    }
}

output {
    csv {
        fields => ["message", "@timestamp", "host"]
        path => "D:\es\logs\export.csv"
    }
}
```

Kafka

Этот плагин предназначен для записи событий в топик Kafka. Он использует API Kafka Producer для записи сообщений в

топик в брокере. Единственная необходимая настройка — это `topic_id`.

Рассмотрим конфигурацию Kafka:

```
#kafka.conf

input {
    stdin{
    }
}

output {
    kafka {
        bootstrap_servers => "localhost:9092"
        topic_id => 'logstash'
    }
}
```

Параметр `bootstrap_servers` собирает список всех серверных подключений в виде `host1:port1, host2:port2`. Эта информация будет использоваться только для получения метаданных (топиков, разделов и копий). Сокет-соединения для отправки фактических данных будут установлены на основе информации брокера, которая возвращается в метаданных. `topic_id` обозначает название темы, в которой будут опубликованы сообщения.



Обратите внимание: только Kafka версии 0.10.0.x совместима с Logstash версий от 2.4.x до 5.x.x и плагином вывода Kafka

версии 5.x.x.

PagerDuty

Этот плагин вывода будет отправлять уведомления, базируясь на предварительно настроенных службах и политиках эскалации оповещения. Единственным обязательным параметром для этого плагина является `service_key` для указания ключа Service API.

Рассмотрим простой пример с базовой конфигурацией PagerDuty. Она настроена таким образом, чтобы Elasticsearch запрашивала в индексе "nginxlogs" все документы, соответствующие `statuscode:404`, и для каждого найденного документа поднимались события `pagerduty`:

```
#kafka.conf

input {
    elasticsearch {
        hosts => "localhost:9200"
        index => "nginxlogs"
        query  => '{ "query": { "match": { "statuscode": 404} }}'
    }
}

output {
    pagerduty {
        service_key => "service_api_key"
        details => {
            "timestamp" => "%{[@timestamp]}"
        }
    }
}
```

```
        "message" => "Problem found: %  
{[message]}"  
    }  
    event_type => "trigger"  
}  
}
```

Плагины кодеков

В следующих разделах мы рассмотрим несколько наиболее распространенных плагинов кодеков.

JSON

Этот кодек полезен в тех случаях, если данные состоят из документов .json. Он предназначен для кодирования (если используется в плагинах вывода) или декодирования (в случаях с плагинами ввода) данных в формате .json. Если направляемые данные представляют собой массив JSON, будут создаваться множественные события (по одному на каждый элемент).

Пример простого применения плагина JSON выглядит следующим образом:

```
input{  
    stdin{  
        codec => "json"  
    }  
}
```



Если у вас несколько записей JSON, разделенных \n, необходимо использовать кодек json_lines.

Если кодек JSON получает данные из ввода, состоящего из некорректных документов JSON, они будут обработаны как простой текст с добавлением тега _jsonparsefailure.

Rubydebug

Этот кодек может выводить ваши события Logstash с применением библиотеки Ruby Awesome Print.

Пример простого применения этого плагина кодека выглядит следующим образом:

```
output{
  stdout{
    codec => "rubydebug"
  }
}
```

Multiline

Этот кодек полезен для слияния нескольких строк данных с единичным событием. Он подойдет и в тех случаях, когда необходимо отследить стек единичного события и информацию, которая разбита по нескольким строкам.

Ниже вы видите образец применения этого кодека:

```
input {
  file {
    path => "/var/log/access.log"
    codec => multiline {
```

```
        pattern => "^\\s "
        negate => false
        what => "previous"
    }
}
}
```

Кодек Multiline соединяет любую строку, начинающуюся с пробела, с предыдущей строкой.

Плагины фильтрации

Поскольку в следующей главе мы будем детально рассматривать различные способы изменения и дополнения данных логов с помощью разных плагинов фильтрации, сейчас мы пропустим связанную с этим информацию.

Узел поглощения данных

До версии Elasticsearch 5.0, если нам требовалась предварительная обработка документов до индексирования, единственным возможным способом сделать это было использовать Logstash или преобразовывать их вручную, а потом индексировать в Elasticsearch. Не хватало функционала для возможности предварительной обработки/трансформации документов, они индексировались в исходном виде. Однако в версии Elasticsearch 5.x и далее была представлена функция узла поглощения данных, которая является легковесным решением для предварительной обработки и дополнения документов в Elasticsearch до индексирования.

Если Elasticsearch запущена в конфигурации по умолчанию, она будет работать как главный узел, узел данных и узел

поглощения. Для отключения функции поглощения измените следующую настройку в файле `elasticsearch.yml`:

```
node.ingest: false
```

Узел поглощения может быть использован для предварительной обработки документов до момента фактического индексирования. Эта обработка происходит путем перехвата составных и индексирующих запросов, применения трансформаций данных и отправки документов обратно к API. С появлением новой функции поглощения Elasticsearch переняла часть функционала Logstash, ответственную за фильтрацию, и теперь мы можем выполнять обработку сырых логов и дополнять их внутри Elasticsearch.

Для предварительной обработки логов перед индексированием необходимо указать контейнер (который выполняет последовательность действий по трансформации входящего документа). Для этого мы попросту указываем параметр `pipeline` в индексе или в составном запросе, чтобы узел поглощения знал, какой контейнер использовать:

```
POST my_index/my_type?pipeline=my_pipeline_id
{
  "key": "value"
}
```

Определение контейнера

Контейнер определяет серию процессоров. Каждый процессор трансформирует документ в том или ином виде. Каждый процессор запускается в том порядке, в котором он определен в контейнере. Контейнер хранит два основных поля: описание и список процессоров. Параметр `description` —

необязательное поле, которое предназначено для хранения информации, связанной с использованием контейнера; с помощью параметра `processors` можно указать список процессоров для трансформации документа.

Типичная структура контейнера выглядит следующим образом:

```
{  
  "description" : "...",  
  "processors" : [ ... ]  
}
```

Узел поглощения имеет около 20 встроенных процессоров, включая `gsub`, `grok`, `convert`, `remove`, `rename` и пр. для использования в создании контейнеров. Вместе со встроенными процессорами вы также можете применять доступные плагины поглощения, такие как `ingest attachment`, `ingest geo-ip`, `ingest user-agent`. По умолчанию они недоступны, но можно установить их как любой другой плагин Elasticsearch.

Ingest API

Узел поглощения предоставляет набор API, известный как Ingest API. Вы можете использовать эти API для определения, симуляции, удаления или поиска информации о контейнерах. Конечная точка данного API — `_ingest`.

API определения контейнера

Этот API используется для определения и добавления нового контейнера, а также для обновления имеющихся контейнеров.

Рассмотрим пример. Как видно в следующем коде, мы определили новый контейнер с названием `firstpipeline`,

который переводит значения поля `message` в верхний регистр:

```
curl -X PUT http://localhost:9200/_ingest/pipeline/firstpipeline -H 'content-type: application/json' -d '{ "description" : "uppercase the incoming value in the message field", "processors" : [ { "uppercase" : { "field": "message" } } ] }'
```

При создании контейнера вы можете указать несколько процессоров, а порядок их выполнения зависит от порядка, в котором они указаны в конфигурации. Рассмотрим это на примере. Как видно в следующем коде, мы создали новый контейнер с именем `secondpipeline`, который преобразует значения поля `"message"` в верхний регистр и переименовывает поле `"message"` в `"data"`. Он создает новое поле с названием `"label"` и значением `testlabel`:

```
curl -X PUT http://localhost:9200/_ingest/pipeline/secondpipeline -H 'content-type: application/json' -d '{
```

```

    "description" : "uppercase the incomming
value in the message field",
    "processors" : [
        {
            "uppercase" : {
                "field": "message",
                "ignore_failure" : true
            }
        },
        {
            "rename": {
                "field": "message",
                "target_field": "data",
                "ignore_failure" : true
            }
        },
        {
            "set": {
                "field": "label",
                "value": "testlabel",
                "override": false
            }
        }
    ]
}

```

Используем второй контейнер для индексирования документа-образца:

```

curl -X PUT
'http://localhost:9200/myindex/mytype/1?
pipeline=secondpipeline'

```

```
curl -X POST "http://localhost:9200/_index/_type/_id" -H  
'content-type: application/json' -d '{  
  "message": "elk is awesome"  
}'
```

А теперь мы получим тот же документ и утвердим трансформацию:

```
curl -X GET "http://localhost:9200/myindex/mytype/1" -H  
'content-type: application/json'
```

Ответ:

```
{  
  "_index": "myindex",  
  "_type": "mytype",  
  "_id": "1",  
  "_version": 1,  
  "found": true,  
  "_source": {  
    "label": "testlabel",  
    "data": "ELK IS AWESOME"  
  }  
}
```



Если отсутствует поле, используемое процессором, тогда процессор сгенерирует исключение и документ не будет проиндексирован. Для предотвращения исключений процессора мы можем присвоить параметру "ignore_failure" значение true.

API получения контейнера

Этот API используется для получения определения существующего контейнера. С его помощью можно найти детали определения одного контейнера или всех.

Команда для поиска определений всех контейнеров выглядит следующим образом:

```
curl -X GET  
http://localhost:9200/_ingest/pipeline -H  
'content-type: application/json'
```

Для поиска определения существующего контейнера введите его идентификатор для получения .api контейнеров. В примере ниже вы можете увидеть поиск определения контейнера с названием secondpipeline:

```
curl -X GET  
http://localhost:9200/_ingest/pipeline/secondpi  
eline -H  
'content-type: application/json'
```

API удаления контейнера

Данный API удаляет контейнеры согласно идентификатору или совпадению с универсальным символом. Далее вы увидите пример удаления контейнера с названием firstpipeline:

```
curl -X DELETE  
http://localhost:9200/_ingest/pipeline/firstpi  
eline -H  
'content-type: application/json'
```

API симуляции контейнера

Используется для симуляции выполнения контейнера относительно набора документов, выбранных в теле запроса. Можно указать как существующий контейнер, так и определение необходимого контейнера. Для симуляции контейнера поглощения добавьте конечную точку `"_simulate"` к API контейнера.

Ниже приведен пример симуляции существующего контейнера:

```
curl -X POST  
http://localhost:9200/_ingest/pipeline/firstpipeline/_simulate -H  
'contenttype: application/json' -d '{  
  "docs" : [  
    { "_source": {"message":"first document"} } ,  
    { "_source": {"message":"second document"} }  
  ]  
}'
```

Далее вы увидите пример симулированного запроса с определением контейнера в теле запроса:

```
curl -X POST  
http://localhost:9200/_ingest/pipeline/_simulate -H  
'contenttype: application/json' -d '{  
  "pipeline" : {  
    "processors": [  
      {  
        "join": {  
          "field": "message",
```

```
        "separator": "-"
    }
}
},
"docs" : [
    {
        "_source": {
            "message": ["first", "document"]
        }
    }
]
```

Резюме

В этой главе мы поговорили об основах Logstash. Мы установили и настроили Logstash, чтобы можно было начать работу с контейнерами данных, а также разобрались в архитектуре Logstash.

Вы узнали об узле поглощения (ingest node), который впервые появился в версии Elastic 5.x и который сегодня можно использовать вместо выделенной установки Logstash. Вы научились применять узел поглощения для предварительной обработки документов до момента фактического индексирования, а также познакомились с различными API.

В следующей главе мы поговорим о разнообразных фильтрах в арсенале Logstash, которые ставят его в один ряд с другими фреймворками потоковой обработки в реальном и псевдореальном времени с нулевым кодированием.

6. Разработка контейнеров данных с помощью Logstash

Прочитав предыдущую главу, вы наверняка поняли важность Logstash в процессе анализа логов. Вы также узнали о способах применения этого компонента и его высокоуровневой архитектуры, а также прошлись по нескольким наиболее распространенным плагинам. Один из важных процессов в Logstash — это структуризация лог-данных, не имеющих четкой структуры. Она облегчает поиск релевантной информации, а также улучшает возможности анализа. Помимо обработки и структуризации лог-данных, полезным будет и их дополнение для улучшения понимания логов. Logstash выполняет эту работу. В предыдущей главе вы узнали, что Logstash может читать данные из широкого спектра источников ввода и что это тяжелый процесс. Установка Logstash на граничных узлах доставки логов не всегда возможна. Есть ли альтернативный легковесный агент, который может использоваться для этой цели? Вы узнаете это в текущей главе.

Мы рассмотрим следующие темы.

- Обработка и дополнение логов с помощью Logstash.
- Платформа Elastic Beats.
- Установка и настройка Filebeats для доставки логов.

Обработка и дополнение логов с помощью Logstash

Структурированные данные проще анализировать по сравнению с неструктуризованными: можно сделать это более

осмысленно/глубоко. Большинство инструментов анализа зависят от структуры данных. Для анализа и визуализации мы будем использовать Kibana, но этот компонент будет работать эффективно, только если данные в Elasticsearch имеют корректный вид (информация лог-данных загружена в соответствующие поля, а тип данных в полях подходящий).

Обычно лог-данные состоят из двух частей:

```
logdata = timestamp + data
```

`timestamp` — это время, когда произошло событие, а `data` — информация о событии. `data` может содержать один фрагмент информации или более. Например, если мы возьмем логи `apache-access`, фрагмент данных будет содержать код ответа, URL запроса, IP-адрес и т.д. Нам понадобится механизм для извлечения этой информации из данных и конвертирования неструктурированных данных/событий в структурированные. Именно здесь пригодится функционал Logstash в виде контейнеров фильтрации. Фильтрационная часть состоит из одного или нескольких плагинов фильтрации, которые помогают обрабатывать и дополнять лог-данные.

Плагины фильтрации

Плагин фильтрации предназначен для преобразования данных. Возможно комбинировать один или несколько плагинов, а порядок их использования определяет порядок, в котором будут преобразованы данные. Пример секции фильтрации в контейнере Logstash будет выглядеть так, как показано на рис. 6.1.

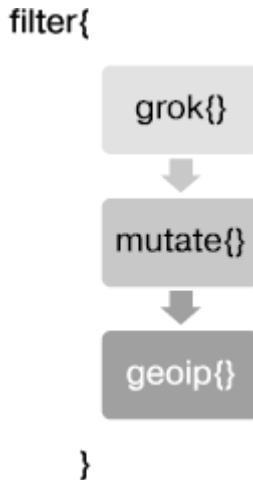


Рис. 6.1

Полученные от плагина ввода события передаются через каждый из плагинов, указанных в секции фильтров, в процессе чего происходит преобразование событий в зависимости от определенных плагинов. В конце выполняется отправка в плагин вывода с целью передачи события в выбранное место.

В следующих разделах мы рассмотрим несколько наиболее популярных плагинов фильтрации, используемых для преобразования данных.

Фильтр CSV

Этот фильтр полезен для обработки файлов .csv. Плагин берет данные CSV, содержащие события, обрабатывает их и сохраняет как отдельные поля.

Взглянем на работу фильтра CSV на примере данных. Сохраните следующие данные в файле с названием users.csv:

```
FName,LName,Age,Salary,EmailId,Gender
John,Thomas,25,50000,John.Thomas,m
```

```
Raj , Kumar , 30 , 5000 , Raj . Kumar , f  
Rita , Tony , 27 , 60000 , Rita . Tony , m
```

Следующий фрагмент кода показывает работу плагина фильтрации CSV. У него нет обязательных параметров. Он сканирует каждую строку данных и применяет стандартные названия столбцов, такие как `column1`, `column2` для размещения данных. По умолчанию данный плагин использует `,` (запятую) как разделитель полей. Разделитель по умолчанию может быть изменен в параметре `separator`. Вы можете указать список названий столбцов либо с помощью параметра `columns`, который принимает массив названий столбцов, либо включив параметр `autodetect_column_names`. Таким образом плагин будет знать, что ему необходимо автоматически определять названия столбцов:

```
#csv_file.conf
input {
    file{
        path => "D:\es\logs\users.csv"
        start_position => "beginning"
    }
}

filter {
    csv{
        autodetect_column_names => true
    }
}

output {
    stdout {
```

```
    codec => rubydebug
}
}
```

Фильтр Mutate

Этот фильтр позволяет выполнять различные изменения данных в полях. Вы можете переименовать, конвертировать, извлекать и модифицировать поля в событиях.

Улучшим конфигурацию `csv_file.conf` из предыдущего раздела с фильтром `mutate` и разберемся, как его применять. На примере следующего кода показано использование фильтра `Mutate`:

```
#csv_file_mutuate.conf
input {
  file{
    path => "D:\es\logs\users.csv"
    start_position => "beginning"
    sincedb_path => "NULL"
  }
}

filter {
  csv{
    autodetect_column_names => true
  }
  mutate {
    convert => {
      "Age" => "integer"
      "Salary" => "float"
    }
  }
}
```

```

rename => { "FName" => "Firstname"
               "LName" => "Lastname" }

gsub => [
    "EmailId", "\.", "_"
]
strip => ["Firstname", "Lastname"]
uppercase => [ "Gender" ]
}

}

output {
  stdout {
    codec => rubydebug
  }
}

```

Как видно из примера, настройка `convert` внутри фильтра помогает изменять тип данных поля. Возможные итоговые конверсии — целое число, строка, число с плавающей точкой и булево значение.



Если тип конверсии — булев, допустимы лишь следующие значения.

Правда: `true`, `t`, `yes`, `y` и `1`.

Ложь: `false`, `f`, `no`, `n` и `0`.

Параметр `rename` внутри фильтра используется для переименования одного или нескольких полей. Предыдущий

пример переименовывает поле FName в Firstname и LName в Lastname.

Параметр `gsub` используется для сопоставления регулярного выражения и значения поля, а также для замены всех соответствий замещающей строкой. Поскольку регулярные выражения работают только со строками, это поле может содержать лишь строки или массивы строк. Берется массив, хранящий три элемента на одно подстановочное значение (иными словами, берется название поля, регулярное выражение и заменяемая строка). В предыдущем примере символ `.` в поле `EmailId` заменен на `_`.



Убедитесь, что при компоновке регулярного выражения вы не используете специальные символы, такие как `\`, `+`, `.` и `?`.

Параметр `strip` используется для извлечения пробелов.



Порядок параметров внутри фильтра `mutate` имеет значение. Поля будут редактироваться в том порядке, в котором указаны параметры. Например, если поля `FName` и `LName` во входящем событии были переименованы в `Firstname` и `Lastname` с помощью параметра `rename`, другие настройки более не могут ссылаться на `FName` и `LName`. Для этого им придется использовать новые, переименованные поля.

Параметр `uppercase` предназначен для преобразования строки в верхний регистр. В предыдущем примере в верхний

регистр преобразовано значение в поле `Gender`.

Аналогичным образом, используя различные параметры фильтра `Mutate`, такие как `lowercase`, `update`, `replace`, `join` и `merge`, вы можете перевести строку в нижний регистр, обновить существующее поле, заменить значение поля, присоединить массив значений или произвести слияние полей.

Фильтр Grok

Это мощный и популярный плагин для преобразования неструктурированных данных в структурированные, для упрощения запросов и фильтрации этих данных. В общих чертах `Grok` позволяет сопоставить строку и шаблон (который основан на регулярном выражении) и отобразить определенные части строки на свои поля. Общий синтаксис шаблона `Grok` выглядит следующим образом:

```
%{PATTERN:FIELDNAME}
```

`PATTERN` — имя шаблона, который соответствует тексту.
`FIELDNAME` — идентификатор для соответствующего фрагмента текста.

По умолчанию поля `Grok` строкового типа. Для выбора значений `float` или `int` необходимо использовать следующий формат:

```
%{PATTERN:FIELDNAME:type}
```

Около 120 шаблонов по умолчанию поставляются с `Logstash`. Они являются гибкими и расширяемыми. На базе существующих вы можете создать свой шаблон. Все шаблоны построены на основе библиотеки регулярных выражений `Oniguruma`.

Шаблоны содержат название и регулярное выражение, например:

```
USERNAME [a-zA-Z0-9._-]+
```

Шаблоны могут включать в себя другие шаблоны. Например:

```
HTTPDATE %{MONTHDAY}/%{MONTH}/%{YEAR}: %{TIME} % {INT}
```

С полным списком шаблонов вы можете ознакомиться по следующей ссылке: <https://github.com/logstash-plugins/logstash-patterns-core/blob/master/patterns/grok-patterns>.

Если шаблон недоступен, можно указать регулярное выражение, используя следующий формат:

```
(?<field_name>regex)
```

Например, regex (?<phone>\d\d\d-\d\d\d\d-\d\d\d\d) будет соответствовать телефонным номерам, таким как 123-123-1234, и вставлять обработанные значения в поле phone.

Взглянем на примеры, чтобы лучше понять работу фильтра Grok:

```
#grok1.conf

input {
  file{
    path => "D:\es\logs\msg.log"
    start_position => "beginning"
    since_db_path => "NULL"
  }
}
```

```

filter {
    grok{
        match   => {"message"      => "%{TIMESTAMP_ISO8601:eventtime}"
                     %{USERNAME:userid}%
{GREEDYDATA:data}" }
    }
}

output {
    stdout {
        codec => rubydebug
    }
}

```

Если строка ввода имеет формат "2017-10-11T21:50:10.000+00:00 tmi_19 001 this is a random message", то вывод будет выглядеть следующим образом:

```

{
    "path" => "D:\\es\\\\logs\\\\msg.log",
"@timestamp" => 2017-11-24T12:30:54.039Z,
    "data" => "this is a random message\r",
    "@version" => "1",
    "host" => "SHMN-IN",
    "messageId" => 1,
        "eventtime"          =>      "2017-10-
11T21:50:10.000+00:00",

    "message" => "2017-10-11T21:50:10.000+00:00
tmi_19 001 this is
                                a random message\r",
    "userId" => "tmi_19"
}
```

}



Если шаблон не соответствует тексту, он добавит тег _grokparsefailure в поле tags.

По ссылке <http://grokdebug.herokuapp.com> вы можете найти инструмент, который помогает строить шаблоны, соответствующие логам.



X-Pack, начиная с версии 5.5, содержит компонент Grok Debugger, который автоматически включается, когда вы устанавливаете X-Pack в Kibana. Он находится на вкладке DevTools (Инструменты разработчика) программы Kibana.

Фильтр даты

Этот плагин используется для обработки даты в полях. Он очень удобен и полезен при работе с временными рядами событий. По умолчанию Logstash добавляет поле @timestamp для каждого события, оно содержит время обработки события. Но пользователю может быть интересно не время обработки, а фактическое время, когда произошло событие. Таким образом, с помощью данного фильтра мы можем обрабатывать метку даты/времени из полей и далее применять ее как метку времени самого события.

Возможно использовать плагин следующим образом:

```
filter {
```

```
date {  
    match => [ "timestamp",  
    "dd/MMM/YYYY:HH:mm:ss Z" ]  
}  
}
```

По умолчанию фильтр даты перезаписывает поле `@timestamp`, но вы можете это изменить, явно указав целевое поле, как показано во фрагменте кода ниже. Пользователь также может оставить время, обработанное Logstash:

```
filter {  
    date {  
        match => [ "timestamp",  
        "dd/MMM/YYYY:HH:mm:ss Z" ]  
        target => "event_timestamp"  
    }  
}
```



Часовой пояс по умолчанию будет соответствовать местному времени сервера, если не указано иное. Чтобы вручную выбрать часовой пояс, используйте в плагине параметр `timezone`. Вы можете найти корректные значения часовых поясов по следующей ссылке: <http://joda-time.sourceforge.net/timezones.html>.

Если поле времени имеет несколько возможных форматов времени, их можно указать как массив значений в параметре `match`:

```
match => [ "eventdate", "dd/MMM/YYYY:HH:mm:ss  
Z", "MMM dd yyyy  
HH:mm:ss", "MMM d yyyy HH:mm:ss", "ISO8601" ]
```

Фильтр Geoip

Этот плагин предназначен для пополнения логов информацией. При наличии IP-адреса будет добавлено его географическое местоположение. Географическая информация будет найдена с помощью поиска корректного IP-адреса по базе данных GeoLite2 City, и результаты будут добавлены в поля. База данных GeoLite2 City является продуктом организации Maxmind и доступна по лицензии CCA-ShareAlike 4.0. Logstash поставляется в комплекте с этой базой данных, следовательно, при поиске не придется писать какие-либо сетевые запросы, что позволяет ускорить поиск.

Единственным обязательным параметром этого плагина является `source`, который принимает IP-адреса в строковом формате. Этот плагин создает поле `geoip` с географическими данными: названием страны, области, города, почтовым кодом и др. Поле `[geoip][location]` создается в случае, если поиск GeoIP возвращает широту и долготу и размечен к типу `geopoint` при индексировании Elasticsearch. Поля `geopoint` могут использоваться для запроса геоформы в Elasticsearch, функций фильтрации, а также для создания визуализации на карте в Kibana (рис. 6.2).

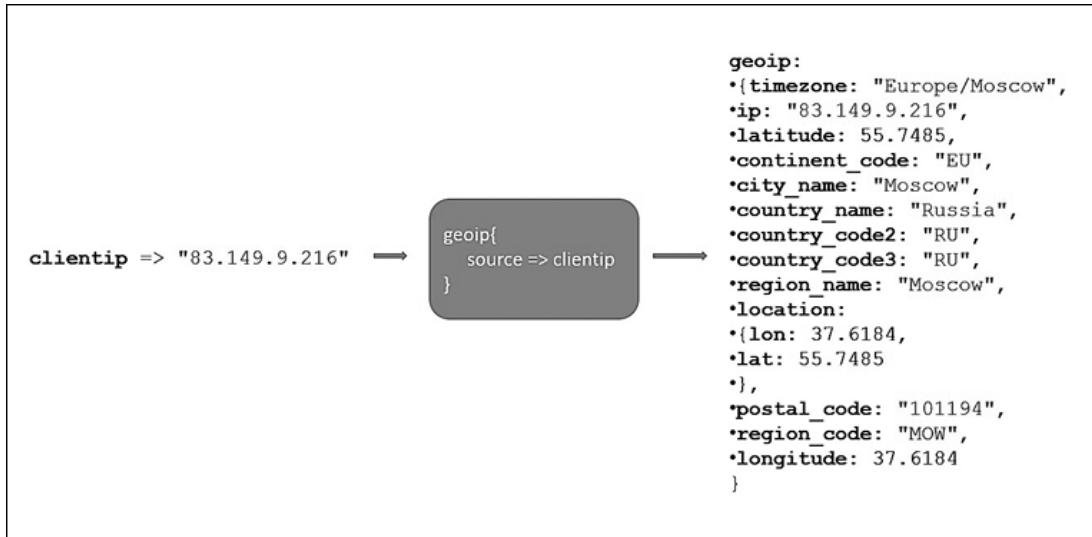


Рис. 6.2



Фильтр Geoip поддерживает поиск по обоим форматам IPv4 и IPv6.

Фильтр Useragent

Этот фильтр преобразует (parses) строки пользовательского агента (user agent) в структурированные данные, базируясь на данных BrowserScope (<http://www.browserscope.org/>). Он добавляет следующую информацию о пользовательском агенте: семейство, операционная система, устройство и т.д. Для извлечения деталей этот фильтр использует базу данных `regexes.yaml`, которая поставляется в комплекте с Logstash. Единственным обязательным параметром для этого плагина является `source`, который принимает строки, содержащие детали о пользовательском агенте (рис. 6.3).



Рис. 6.3

Введение в Beats

Фреймворк *Beats* представляет собой легковесные компоненты для доставки данных, которые устанавливаются как агенты на пограничные серверы для передачи операционных данных в Elasticsearch. Как и Elasticsearch, Logstash и Kibana, Beats является продуктом с открытым исходным кодом. В зависимости от условий применения вы можете настроить Beats для доставки данных в Elasticsearch и для преобразования событий до их передачи в Elasticsearch.

Фреймворк Beats создан на основе библиотеки *Libbeat*, которая предоставляет инфраструктуру для упрощения процесса доставки операционных данных в Elasticsearch. Она предлагает API, которые могут быть использованы всеми

компонентами Beats для отправки данных в источник вывода (такие как Elasticsearch, Logstash, Redis, Kafka и пр.), настроек параметров ввода/вывода, обработки событий, внедрения сбора данных и пр. Библиотека Libbeat создана на основе языка программирования Go. Он был выбран по причине простоты обучения, нетребовательности к ресурсам, статичной компиляции и легкого развертывания.

Компания Elastic.co разработала и поддерживает несколько компонентов Beats, таких как Filebeat, Packetbeat, Metricbeat, Heartbeat и Winlogbeat. Есть также несколько компонентов, созданных сообществом, в том числе amazonbeat, mongobeat, httpbeat и nginxbeat, которые были встроены в фреймворк Beats. Некоторые из компонентов расширены для соответствия нуждам бизнеса или имеют точки расширения. Если вы не нашли компонент Beats для своих целей, можете создать собственный с помощью библиотеки Libbeat (рис. 6.4).

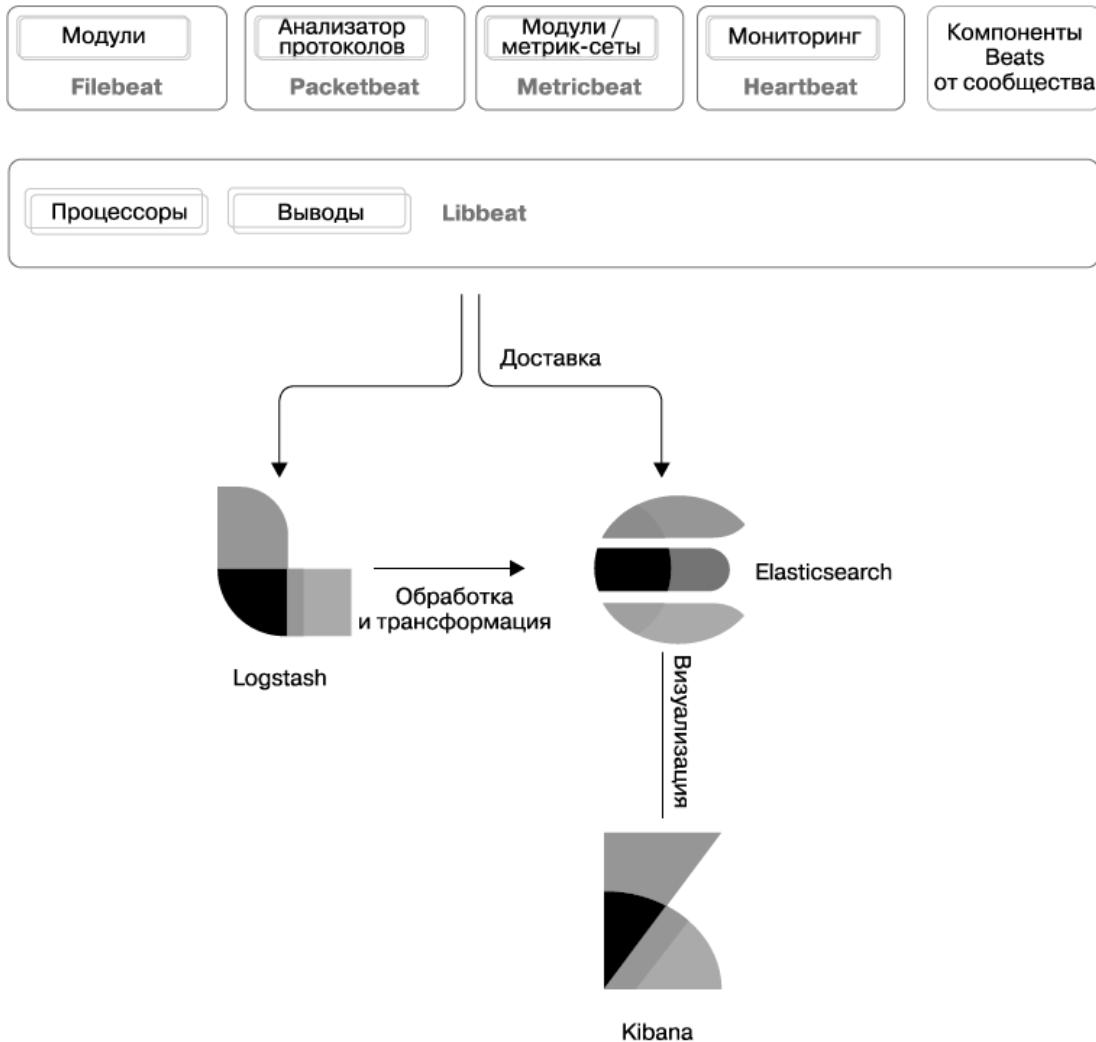


Рис. 6.4

Beats от Elastic.co

В следующих разделах мы рассмотрим некоторые из широко распространенных компонентов Beats от Elastic.co.

Filebeat

Это легковесный агент доставки логов из локальных файлов, работает с открытым исходным кодом. Filebeat выступает как бинарный компонент, для запуска не нужна среда выполнения

наподобие JVM, следовательно, он очень легковесный, исполняемый и потребляет меньше ресурсов. Filebeat устанавливается как агент на пограничных серверах, с которых необходимо доставлять логи. Он мониторит папки с логами или конкретные лог-файлы и направляет их в Elasticsearch, Logstash, Redis или Kafka. Агент легко масштабируется и предоставляет возможность доставлять логи из нескольких систем в централизованную систему/на сервер, из которых логи могут быть обработаны.

Metricbeat

Metricbeat — это легковесный поставщик для периодического сбора метрик из операционных систем и сервисов, запущенных на ваших серверах. Он помогает мониторить систему путем сбора метрик из нее, а также таких сервисов, как Apache, MondoDB, Redis и пр., запущенных на выбранном сервере. Metricbeat может передавать собранные метрики напрямую в Elasticsearch или отправлять в Logstash, Redis или Kafka. Для мониторинга сервисов Metricbeat может быть установлен на пограничный сервер, на котором запущены сервисы; также есть возможность сбора метрик и с удаленного сервера. Тем не менее рекомендуется устанавливать Metricbeat на тех серверах, на которых запущены сервисы.

Packetbeat

Packetbeat представляет собой анализатор сетевых пакетов в реальном времени, который работает путем захвата сетевого трафика между серверами приложений, декодируя протоколы уровня приложений (HTTP, MySQL, Redis, Memcache и др.), корректируя запросы ответами и записывая конкретные поля для каждой транзакции. Packetbeat отслеживает трафик между

серверами, обрабатывает протоколы уровня приложения на лету и преобразует сообщения в транзакции. С его помощью можно легко диагностировать проблемы с приложением, например баги или проблемы производительности, и облегчить устранение неполадок. Packetbeat может быть запущен на том же сервере, где запущено приложение, или на отдельном. Собранная информация о транзакциях отправляется в настроенный вывод, а именно в Elasticsearch, Logstash, Redis или Kafka.

Heartbeat

Heartbeat — это новое дополнение в экосистеме Beats, которое используется для проверки того, работает ли сервис и доступен ли он. Heartbeat является легковесным демоном, который устанавливается на удаленном сервере для периодических проверок статуса работы сервисов, запущенных на сервере. Heartbeat поддерживает мониторинг ICMP, TCP и HTTP для проверки хостов/сервисов.

Winlogbeat

Winlogbeat специализируется на платформе Windows. Он устанавливается в виде службы Windows в операционной системе Windows XP или новее для чтения из одного или нескольких журналов событий с использованием API Windows. Он фильтрует события согласно настроенным пользователем параметрам, после чего отправляет данные событий на выбранный вывод, например в Elasticsearch или Logstash.

Winglobeat может захватывать такие данные, как события приложений, события оборудования, события безопасности и события системы.

Auditbeat

Auditbeat — это новое дополнение к семье компонентов Beats; впервые появился в версии Elastic Stack 6.0. Это легковесный компонент доставки данных, который устанавливается на серверах для мониторинга активности пользователей и процессов, а также для анализа данных событий в Elastic Stack без запуска Linux auditd. Компонент Auditbeat взаимодействует напрямую с фреймворком Linux audit, собирает аналогичный набор данных и в реальном времени отправляет данные в Elastic Stack. Кроме того, с его помощью вы можете следить за изменениями в определенных папках. Информация обо всех изменениях файлов отправляется в выбранный источник вывода в реальном времени, таким образом возможна идентификация потенциальных нарушений политики безопасности.

Компоненты Beats от сообщества

Разработчики из сообщества открытого исходного кода создают свои компоненты Beats, используя соответствующий фреймворк. В табл. 6.1 перечислено несколько таких компонентов.

Таблица 6.1

Название компонента Beats	Описание
springbeat	Собирает сведения о метриках и рабочем состоянии из приложений Spring Boot, запущенных в пределах исполнительного модуля
rsbeat	Доставляет логи redis в Elasticsearch
nginxbeat	Читает статусы в Nginx
mysqlbeat	Выполняет запрос в MySQL и отправляет результаты в Elasticsearch

Название компонента Beats	Описание
mongobeat	Мониторит процессы MongoDB и может отправлять различные типы документов в Elasticsearch в зависимости от конфигурации
gabeat	Собирает данные из API Google Analytics в реальном времени
apachebeat	Читает статусы на серверах Apache HTTPD
amazonbeat	Считывает данные из выбранного продукта Amazon

Полный список компонентов Beats от сообщества вы можете найти по следующей ссылке: <https://www.elastic.co/guide/en/beats/devguide/current/community-beats.html>.



Имейте в виду, что компания Elastic.co не несет ответственности за компоненты Beats от сообщества, равно как и не осуществляет их поддержку.



Руководство разработчика Beats предоставляет необходимую информацию для создания собственного компонента. Руководство можно найти по следующей ссылке: <https://www.elastic.co/guide/en/beats/devguide/current/index.html>.

Logstash против Beats

Возможно, вы в небольшом замешательстве после прочтения предыдущих разделов с вводной информацией о компонентах

Logstash и Beats, ведь не совсем понятно, взаимозаменяемы ли они и в чем между ними разница, в каких ситуациях использовать тот или иной компонент. Beats — это легковесные агенты, их особенность заключается в задействовании небольшого количества ресурсов. Следовательно, они устанавливаются на тех пограничных серверах, с которых необходимо собирать операционные данные. У компонента Beats нет мощного функционала Logstash в плане обработки и трансформации событий. Logstash имеет широкий ассортимент плагинов ввода, вывода, фильтрации, целью которых являются сбор, дополнение и преобразование данных из различных источников. Однако он требователен к ресурсам системы и также может использоваться как независимый продукт вне комплекса Elastic Stack. Рекомендуется устанавливать Logstash на выделенный сервер, а не на пограничные. Компоненты Beats и Logstash являются взаимодополняемыми и в зависимости от целей можно использовать их оба вместе либо по отдельности, как это описано в разделе «Введение в Beats».

Filebeat

Filebeat — это легковесный компонент с открытым исходным кодом, который устанавливается как агент для доставки логов из локальных файлов. Он мониторит определенные папки на наличие лог-файлов, отслеживает их и перенаправляет данные в Elasticsearch, Logstash, Redis или Kafka. Он легко масштабируется и способен доставлять логи из различных систем на централизованный сервер/в систему, где логи могут быть обработаны.

Скачивание и установка Filebeat

Перейдите по ссылке <https://www.elastic.co/downloads/beats/filebeat> и скачайте файл формата .tar/.zip в зависимости от вашей операционной системы. Установка компонента Filebeat довольно проста и понятна (рис. 6.5).



Рис. 6.5



Версии Beats 6.0.x совместимы с Elasticsearch 5.6.x и 6.0.x и Logstash версий 5.6.x и 6.0.x. Полную схему совместимости вы можете найти по ссылке: https://www.elastic.co/support/matrix#matrix_compatibility. Прежде чем рассматривать примеры использования Elasticsearch и Logstash совместно с Beats в этой главе, убедитесь, что у вас установлены совместимые версии.

Установка под Windows

Распакуйте скачанный файл. После распаковки перейдите в созданную папку, как показано в следующем фрагменте кода:

```
D:> cd D:\packt\filebeat-6.0.0-windows-x86_64
```



Для установки Filebeat как службы в Windows выполните следующие шаги.

1. Откройте Windows PowerShell как администратор и перейдите в папку с распакованными файлами.
2. Из командной строки PowerShell выполните следующие команды для установки Filebeat как службы Windows:

```
PS >cd D:\packt\filebeat-6.0.0-windows-x86_64  
PS D:\packt\filebeat-6.0.0-windowsx86_64>.\install-service-filebeat.ps1
```

Если в вашей системе отключено выполнение скриптов, необходимо настроить нужные политики для текущей сессии таким образом, чтобы вышеописанный скрипт был успешно выполнен. Например:

```
PowerShell.exe -ExecutionPolicy UnRestricted -File .\install-service-filebeat.ps1.
```

Установка под Linux

Распакуйте пакет `tar.gz` и перейдите в созданный каталог, как показано ниже:

```
$> tar -xzf filebeat-6.0.0-linux-x86_64.tar.gz  
$> cd filebeat
```



Для установки с помощью `deb` или `rpm` выполните соответствующие команды в терминале:

`deb`:

```
curl -L -O  
https://artifacts.elastic.co/downloads/beat  
s/filebeat/fil  
ebeat-6.0.0-amd64.deb  
sudo dpkg -i filebeat-6.0.0-amd64.deb
```

`rpm`:

```
curl -L -O  
https://artifacts.elastic.co/downloads/beat  
s/filebeat/fil  
ebeat-6.0.0-x86_64.rpm  
sudo rpm -vi filebeat-6.0.0-x86_64.rpm
```

Filebeat будет установлен в папку `/usr/share/filebeat`. Файлы конфигурации расположены в папке `/etc/filebeat`. Скрипт `init` будет находиться в `/etc/init.d/filebeat`. Файлы `log` будут в папке `/var/log/filebeat`.

Архитектура

Filebeat состоит из трех ключевых компонентов, которые называются *старателями* (prospectors), *сборщиками* (harvesters) и *спулерами* (spooler). Эти компоненты работают совместно для отслеживания файлов и отправки данных событий в источник вывода согласно вашему выбору. Старатель отвечает за идентификацию списка файлов, из которых следует читать логи. В его настройках указываются один или несколько путей, по которым он находит файлы для чтения логов; для каждого файла запускается отдельный сборщик. Сборщик отвечает за чтение содержимого файла. Он читает каждый файл строка за строкой и отправляет содержимое в вывод. Он также отвечает за открытие и закрытие каждого файла, следовательно, во время его работы дескриптор файла остается открытым. Далее сборщик отправляет считанное содержимое (то есть события) спулеру, который занимается агрегацией и отправкой данных в настроенный вывод.

Каждый запущенный процесс Filebeat может быть настроен для работы с одним или несколькими старателями. В данный момент предусмотрена поддержка двух типов старателей: `log` и `stdin`. Если тип ввода — `log`, старатель находит все файлы на диске, которые соответствуют выбранным путям для файлов, и запускает сборщики для каждого файла. Каждый старатель работает по своему маршруту Go. Если тип ввода `stdin`, старатель будет считывать по стандартным вводам.

При каждом чтении файла в Filebeat сборщик запоминает последнюю позицию, и, если прочитанная строка отправлена в вывод, данные о ней сохраняются в файле реестра, который периодически выгружается на диск. Если нет доступа к источнику вывода (такому как Elasticsearch, Logstash, Redis или Kafka), текущая позиция чтения файла запоминается в Filebeat и чтение продолжится сразу же, как только станет доступен

источник вывода. Пока Filebeat работает, информация о позиции хранится в памяти каждого старателя. Если Filebeat был перезапущен, для продолжения работы каждого сборщика будет использована информация о последней обработанной позиции из файла реестра.

Filebeat не считает строку лога доставленной до той поры, пока источник вывода не был опрошен и пока состояние доставки строк в выбранный вывод не записано в файл реестра. Filebeat гарантирует, что события будут доставлены в настроенный вывод как минимум один раз, без потери данных.



Место хранения файла реестра будет следующим: `data/registry` для архивов `.tar.gz` и `.zip`, `/var/lib/filebeat/registry` для пакетов DEB и RPM, `C:\ProgramData\filebeat\registry` для файлов `.zip` Windows, если Filebeat установлен как служба (рис. 6.6).

Более детальную информацию об этом вы найдете по ссылке:

<https://www.elastic.co/guide/en/beats/filebeat/6.0/images/filebeat.png>.

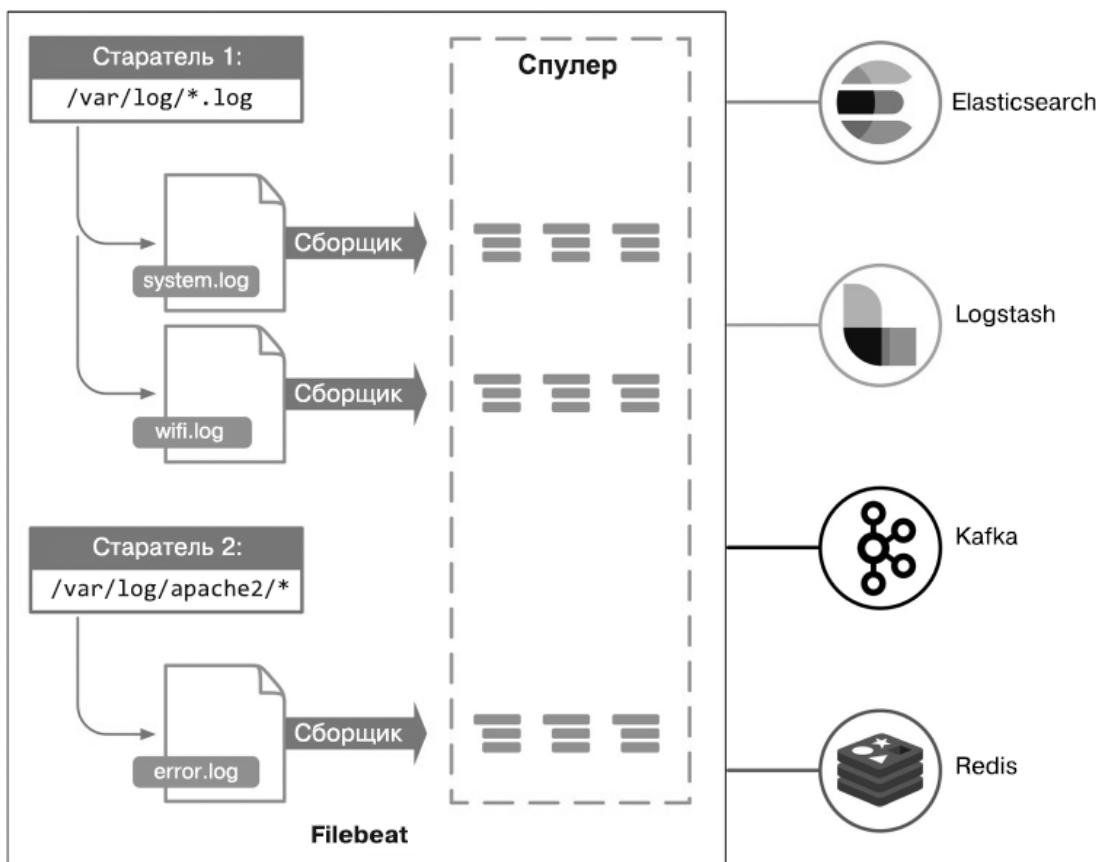


Рис. 6.6

Настройка Filebeat

Все настройки, связанные с Filebeat, хранятся в конфигурационном файле с названием `filebeat.yml`. Он использует синтаксис YAML.

Этот файл содержит разделы с информацией о таких показателях, как:

- старатели;
- глобальные настройки;
- общие настройки;

- конфигурация вывода;
- конфигурация обработки;
- конфигурация папок;
- конфигурация модулей;
- конфигурация панели управления;
- конфигурация сбора данных.



Файл `filebeat.yml` будет присутствовать в папке установки только в том случае, если использовались файлы `.zip` или `.tar`. Если для установки были запущены файлы `DEB` или `RPM`, тогда он будет расположен в каталоге `/etc/filebeat`.

Некоторые из этих разделов являются общими для всех типов компонентов Beats. Прежде чем мы рассмотрим их детально, взглянем на пример простой конфигурации. В примере ниже видно, что при запуске Filebeat ищет файлы с расширением `.log` в папке `D:\packt\logs\`. Он будет доставлять все записи каждого файла в Elasticsearch, который настроен как источник вывода и находится по адресу `localhost:9200`:

```
#filebeat.yml
=====
prospectors
=====
```

Filebeat

```
filebeat.prospectors:  
- input_type: log  
  
# Paths that should be crawled and fetched.  
Glob based paths.  
paths:  
- D:\packt\logs\*.log  
  
#===== Outputs  
=====  
  
#----- Elasticsearch  
output -----  
----  
output.elasticsearch:  
# Array of hosts to connect to.  
hosts: ["localhost:9200"]
```



При внесении изменений в файл filebeat.yml необходимо перезапустить компонент Filebeat, чтобы изменения вступили в силу.

Рекомендуется тестировать конфигурацию каждый раз при внесении изменений. Для тестирования используйте флаг -configtest:

```
D:\packt\filebeat-6.0.0-windows-  
x86_64>filebeat.exe -configtest  
filebeat.yml
```

Config OK



Для указания флагов Filebeat должен быть запущен в фоновом режиме. Невозможно указывать флаги командной строки, если Filebeat запущен с помощью скрипта init.d в виде deb или rpm или в виде службы Windows.

Разместите несколько лог-файлов в папке D:\packt\logs\. Чтобы Filebeat приступил к доставке логов, выполните следующую команду:

Windows:

```
D:\packt\filebeat-6.0.0-windows-x86_64>filebeat.exe
```

Linux:

```
[locationOfFilebeat]$ ./filebeat
```

Для проверки успешной доставки логов в Elasticsearch выполните такую команду:

```
D:\packt>curl -X GET http://localhost:9200/filebeat*/_search?pretty
```

Образец ответа:

```
{  
  "took" : 1,  
  "timed_out" : false,  
  "_shards" : {  
    "total" : 5,
```

```
    "successful" : 5,
    "failed" : 0
  },
  "hits" : {
    "total" : 6,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "filebeat-2017.11.23",
        "_type" : "doc",
        "_id" : "AV_niRjbPYptcfAHfGNx",
        "_score" : 1.0,
        "_source" : {
          "@timestamp" : "2017-11-23T06:20:36.577Z",
          "beat" : {
            "hostname" : "SHMN-IN",
            "name" : "SHMN-IN",
            "version" : "6.0.0"
          },
          "input_type" : "log",
          "message" : "line2",
          "offset" : 14,
          "source" : "D:\\packt\\logs\\test.log",
          "type" : "log"
        }
      },
      {
        "_index" : "filebeat-2017.11.23",
        "_type" : "doc",
```

```
        "_id" : "AV_niRjbPYptcfAHfGNy",
        "_score" : 1.0,
        "_source" : {
            "@timestamp" : "2017-11-
23T06:20:36.577Z",
            "beat" : {
                "hostname" : "SHMN-IN",
                "name" : "SHMN-IN",
                "version" : "6.0.0"
            },
            "input_type" : "log",
            "message" : "line3",
            "offset" : 21,
            "source" : "D:\\packt\\logs\\test.log",
            "type" : "log"
        }
    },
    ...
    ...
    ...
}
```



Доставленные логи будут размещены в индексе filebeat с меткой времени, которая создана по шаблону filebeat-YYYY.MM.DD. Лог-данные будут расположены в поле message.

Для запуска Filebeat при установке deb или rpm выполните команду sudo service filebeat start. Если он установлен в виде

службы Windows, используйте следующую команду в PowerShell: PS C:\> Start-Service filebeat.

Старатели Filebeat

Этот раздел содержит список старателей, которые используются в Filebeat для обнаружения и обработки лог-файлов. Каждая запись начинается с дефиса (-) и содержит определенные для каждого старателя параметры, включая один или несколько путей для поиска файлов.

Образец конфигурации выглядит так, как показано на рис. 6.7.

Возможны следующие настройки для старателей.

- `input_type` — этот параметр принимает типы `log` или `stdin`. Тип `log` используется для чтения каждой строки логов из файла, а тип `stdin` — для чтения из стандартного источника ввода. Типом по умолчанию является `log`.
- `paths` применяется для указания одного или нескольких путей с целью поиска файлов для обработки. Каждый путь указывается с новой строки, начиная с дефиса (-). Параметр `paths` принимает пути на базе Golang glob, равно как и остальные шаблоны этого типа (более детально — по ссылке <https://golang.org/pkg/path/filepath/#Glob>).
- `exclude_lines` — принимает список регулярных выражений для соответствия. Он исключает из списка все строки, содержащие соответствия регулярному выражению. В предыдущем примере конфигурации исключаются все строки, которые начинаются с `DBG`.

```
===== Filebeat prospectors =====

filebeat.prospectors:

- input_type: log
  paths:
    - /var/log/*.log
    - /var/log/messages
  # Exclude lines.
  exclude_lines: ["^DBG"]
  # Include lines.
  include_lines: ["^ERR", "^WARN"]

  tags: ["java_logs"]

  fields:
    env: staging
  #### Multiline options
  multiline.pattern: '^[[\s\S]]*'
  multiline.negate: false
  multiline.match: after

  scan_frequency: 1s

- input_type: log
  paths:
    - /var/log/apache/httpd-*.*.log
  document_type: apache
```

Рис. 6.7

- `include_lines` — параметр принимает список регулярных выражений для соответствия. Он экспортирует из списка все строки, содержащие соответствия регулярному выражению. В предыдущем примере конфигурации экспортируются все строки, которые начинаются с `ERR` или `WARN`.



Регулярные выражения (`regex`) создаются на основе RE2 (более детально — по ссылке: <https://godoc.org/regexp/syntax>). На

данном сайте вы сможете найти все поддерживаемые шаблоны `regex`.

- `tags` – принимает список тегов, которые будут включены в поле `tags` каждого события, которое доставляется Filebeat. `tags` помогает с условной фильтрацией событий в Kibana или Logstash. В предыдущем примере конфигурации в список `tags` добавляется `java_logs`.
- `fields` – используется для указания optionalных полей, которые необходимо включать в каждое событие, которое доставляется Filebeat. По аналогии с `tags` этот параметр помогает с условной фильтрацией событий в Kibana или Logstash. Поля могут содержать скалярные значения, массивы, словари или любые комбинации из перечисленного. По умолчанию все указанные поля будут сгруппированы в словарь `fields` в документе вывода. В предыдущем примере конфигурации новое поле с названием `env` и значением `staging` будет создано под `fields`.



Для того чтобы сохранить поля как высокоуровневые, установите для параметра `fields_under_root` значение `true`.

- `scan_frequency` – применяется для указания интервала времени, после которого старатель проверяет наличие новых файлов в настроенных папках. В предыдущем примере конфигурации старатель проверяет наличие новых

файлов каждую секунду. По умолчанию в параметре `scan_frequency` установлено 10 с.

- `document_type` — используется для указания типа индекса в выводе в Elasticsearch. Тип по умолчанию — `log`. В предыдущем примере конфигурации логи Apache настроены на тип `apache` таким образом, что во время индексирования в Elasticsearch их можно найти под типом `apache`. Название индекса и далее будет соответствовать шаблону `filebeat-YYYY.MM.DD`.
- `multiline` — указывает, как необходимо обрабатывать логи, разделенные по нескольким строкам. Это очень полезно для процессинга сообщений трассировки или исключений стека.

Параметр `pattern` определяет шаблон регулярного выражения для соответствия: `negate` указывает, следует ли отвергать шаблон, и `match`, который задает, как Filebeat сопоставляет соответствующие строки с событием. Параметр `negate` может иметь значение `true` либо `false`; по умолчанию установлено `false`. Параметр `match` может иметь значение `after` либо `before`. В предыдущем примере конфигурации все последующие строки, начинающиеся с пробела, объединяются с предыдущей строкой, которая не начинается с пробела.



Настройка `after` похожа на мультистроковую настройку `previous` в Logstash, равно как и `before` схожа с `next`.

Глобальные настройки Filebeat

Этот раздел содержит настройки конфигурации, которые контролируют поведение Filebeat на глобальном уровне. Ниже приведены некоторые из них.

- `registry_file` — используется для указания местонахождения файла реестра, который применяется для хранения информации о файлах, например о месте чтения и статусе получения считанных строк в настроенных источниках вывода. По умолчанию местонахождение файла реестра таково: `${path.data}/registry`:

```
filebeat.registry_file:  
/etc/filebeat/registry
```



В качестве значения этого параметра допускается указать относительный или абсолютный путь. Если используется относительный путь, считается, что он относится к настройке `${path.data}`.

- `shutdown_timeout` — эта настройка указывает, как долго Filebeat будет ожидать окончания публикации при завершении работы. Если завершение работы Filebeat происходит во время процесса передачи событий, он не будет ждать, пока источник вывода посчитает все события принятыми. Эта настройка дает Filebeat указание ожидать определенный промежуток времени, прежде чем завершать работу. Это выглядит следующим образом:

```
filebeat.shutdown_timeout: 10s
```

Общие настройки Filebeat

Этот раздел содержит настройки конфигурации и некоторые общие настройки, контролирующие поведение Filebeat.

- `name` — название отправителя, который публикует сетевые данные. По умолчанию для этого поля используется `hostname`:

```
name: "dc1-host1"
```

- `tags` — список тегов, который будет включен в поле `tags` для каждого события, которое доставляется с помощью Filebeat. Благодаря тегам можно легко группировать серверы по различным логическим свойствам, а также легко фильтровать события в Kibana и Logstash:

```
tags: ["staging", "web-tier", "dc1"]
```

- `max_procs` — максимальное количество центральных процессоров (ЦП), которые могут быть использованы одновременно. По умолчанию задействуются все логические ЦП, доступные в системе:

```
max_procs: 2
```

Конфигурация вывода

Этот раздел используется для настройки выводов, в которые должны доставляться события. Возможна отправка в один или

несколько выводов одновременно. Источниками вывода являются Elasticsearch, Logstash, Redis, Kafka, файл или консоль.

Некоторые из возможных источников вывода могут быть настроены, как показано ниже.

- `elasticsearch` — используется для отправки событий напрямую в Elasticsearch.

Образец конфигурации вывода в Elasticsearch выглядит следующим образом:

```
output.elasticsearch:  
    enabled: true  
    hosts: ["localhost:9200"]
```

С помощью настройки `enabled` можно включать или выключать вывод. Параметр `hosts` принимает один или несколько узлов/серверов Elasticsearch. Множественные хосты могут быть указаны с целью достижения устойчивости при сбоях. В случаях, когда настроено несколько хостов, события доставляются по этим узлам в циклическом порядке. Если настроена авторизация Elasticsearch, данные аутентификации могут быть введены с помощью параметров `username` и `password`:

```
output.elasticsearch:  
    enabled: true  
    hosts: ["localhost:9200"]  
    username: "elasticuser"  
    password: "password"
```

При доставке событий в контейнер узла поглощения Elasticsearch информацию о контейнере можно указать под

параметром `pipeline` — тогда их можно будет предварительно обработать до хранения в Elasticsearch:

```
output.elasticsearch:  
  enabled: true  
  hosts: ["localhost:9200"]  
  pipeline: "apache_log_pipeline"
```

- `logstash` — используется для отправки событий в Logstash.



Чтобы задействовать Logstash как источник вывода, необходимо настроить его с плагином ввода Beats для получения входящих событий Beats.

Образец конфигурации вывода в Logstash выглядит следующим образом:

```
output.logstash:  
  enabled: true  
  hosts: ["localhost:5044"]
```

С помощью настройки `enabled` можно включать или выключать вывод. Параметр `hosts` принимает один или несколько узлов/серверов Logstash. Множественные хосты могут быть указаны с целью достижения устойчивости при сбоях. Если настроенный хост не отвечает, тогда события будут отправлены в один из других перечисленных в настройках хостов. В случаях, когда настроено несколько хостов, события доставляются по этим узлам в случайном порядке. Для включения балансировки нагрузки событий на

хосты Logstash присвойте параметру `loadbalance` значение `true`:

```
output.logstash:  
    hosts:      ["localhost:5045",  
    "localhost:5046"]  
    loadbalance: true
```

- `console` – применяется для отправки событий в `stdout`. События будут записаны в формате JSON. Полезно для задач тестирования или отладки.

Образец конфигурации консоли выглядит следующим образом:

```
output.console:  
    enabled: true  
    pretty: true
```

Модули Filebeat

Модули Filebeat упрощают процесс сбора, обработки и визуализации логов популярных форматов.

Модуль состоит из одного или нескольких наборов файлов. В набор файлов входят следующие составляющие.

- Конфигурации старателя Filebeat, содержащие папки по умолчанию, в которых следует искать логи. Предоставляется также конфигурация для комбинирования мультистроковых событий при необходимости.
- Определение контейнера поглощения Elasticsearch для обработки и дополнения логов.

- Шаблоны Elasticsearch с определениями полей, используемых с целью корректной разметки полей для событий.
- Примеры панелей управления Kibana, которые могут быть задействованы для визуализации логов.



Для работы модулей Filebeat необходим узел поглощения Elasticsearch, а версия Elasticsearch должна быть не ниже 5.2.

Ниже перечислены модули по умолчанию, которые поставляются в комплекте с Filebeat:

- Apache2;
- Auditd;
- MySQL;
- Nginx;
- Redis;
- Icinga;
- Модуль системы.

Каталог `modules.d` содержит конфигурации по умолчанию для всех модулей, доступных в Filebeat. Конфигурация каждого модуля хранится в файле `.yml`, а название файла соответствует

названию модуля. Например, конфигурация модуля `redis` будет храниться в файле `redis.yml`.

Поскольку каждый модуль поставляется с конфигурацией по умолчанию, вы можете внести необходимые изменения в файле конфигурации модуля.

Базовая конфигурация для модуля `redis` будет выглядеть следующим образом:

```
#redis.yml
- module: redis
  # Main logs
  log:
    enabled: true

    # Set custom paths for the log files. If
    left empty,
    # Filebeat will choose the paths depending
    on your OS.
    #var.paths:    ["/var/log/redis/redis-
server.log*"]

    # Slow logs, retrieved via the Redis API
    # (SLOWLOG)
    slowlog:
      enabled: true

    # The Redis hosts to connect to.
    #var.hosts: ["localhost:6379"]

    # Optional, the password to use when
    connecting to Redis.
    #var.password:
```

Для включения модулей выполните команду `modules enable`, добавив одно или несколько названий модулей:

Windows :

```
D:\packt\filebeat-6.0.0-windows-x86_64>filebeat.exe modules enable redis mysql
```

Linux :

```
[locationOfFileBeat]$./filebeat modules enable redis mysql
```



Если модуль выключен, конфигурация для него в папке `modules.d` будет храниться с расширением файла `.disabled`.

Аналогичным образом для выключения модулей выполните команду `modules disable`, добавив одно или несколько названий модулей. Например:

Windows :

```
D:\packt\filebeat-6.0.0-windows-x86_64>filebeat.exe modules disable redis mysql
```

Linux :

```
[locationOfFileBeat]$./filebeat modules disable redis mysql
```

Как только модуль активирован, выполните команду `setup`, чтобы загрузить рекомендованный шаблон индекса для записи в Elasticsearch и развернуть примеры панелей управления для визуализации данных в Kibana. Это происходит следующим образом:

Windows:

```
D:\packt\filebeat-6.0.0-windows-x86_64>filebeat.exe -e setup
```

Linux:

```
[locationOfFileBeat]$./filebeat -e setup
```

Флаг `-e` указывает вывод в `stdout`. Как только модули активизированы и выполнена команда `setup`, запустите Filebeat, как обычно, для загрузки шаблонов индекса и примеров панелей управления. Компонент приступит к доставке логов в Elasticsearch.



Вам необходимо выполнять команду `setup` во время установки или обновления Filebeat или после активизации нового модуля.

Некоторые модули имеют плагины зависимости, например `ingest-geoip` и `ingest-user-agent`. Для корректной работы их необходимо установить на Elasticsearch до настройки модулей, в противном случае установка завершится ошибкой.

Вместо того чтобы активизировать модули путем ввода их как параметров командной строки, вы можете включать их внутри самого конфигурационного файла `filebeat.yml` и запускать Filebeat, как обычно:

```
filebeat.modules:  
- module: nginx  
- module: mysql
```

Каждый модуль имеет назначенные ему наборы файлов, которые содержат определенные переменные. Их можно заменить своими, с помощью файла конфигурации или параметров командной строки, используя флаг -M во время работы Filebeat.

Если вы работаете с конфигурационным файлом, сделайте следующее:

```
filebeat.modules:  
- module: nginx  
  access:  
    var.paths: ["C:\nginx\access.log*"]
```

Если вы используете командную строку, сделайте следующее:

Windows:

```
D:\packt\filebeat-6.0.0-windows-  
x86_64>filebeat.exe -e -modules=nginx -M  
          "nginx.access.var.paths=  
[C:\nginx\access.log*]"
```

Linux:

```
[locationOfFileBeat]$ ./filebeat -e -  
modules=nginx -M  
          "nginx.access.var.paths=  
[\var\nginx\access.log*]"
```

Резюме

Из этой главы вы узнали о мощных возможностях фильтрации Logstash, которые предназначены для обработки и дополнения событий. Мы также рассмотрели некоторые из наиболее

распространенных плагинов фильтрации. Помимо этого, мы поговорили о работе фреймворка Beats, провели обзор различных компонентов Beats, таких как Filebeat, Heartbeat, Packetbeat и др., а в компоненте Filebeat разобрались особенно детально.

В следующей главе мы обсудим различные возможности X-Pack — коммерческого предложения от Elastic.co, которое содержит такой функционал, как настройки безопасности для Elastic Stack, возможности мониторинга, рассылки уведомлений, построения графиков, создания отчетности и многое другое.

7. Визуализация данных в Kibana

Kibana — это веб-инструмент для аналитики и визуализации, созданный на базе открытого исходного кода. Он позволяет визуализировать данные, которые хранятся в Elasticsearch в различном виде: в таблицах, картах, графиках. Благодаря простому интерфейсу этого компонента вы сможете легко исследовать большие объемы данных и выполнять продвинутую аналитику данных в реальном времени.

В книге мы рассмотрим различные компоненты Kibana и поговорим о том, как использовать их для анализа данных.

В этой главе мы разберем следующие темы.

- Скачивание и установка Kibana.
- Исследование данных с помощью Kibana.
- Визуализации в Kibana.
- Анализ временных данных в Kibana.
- Настройка и разработка известных плагинов в Kibana.

Скачивание и установка Kibana

Как и другие компоненты Elastic Stack, Kibana довольно просто скачать и установить.

Перейдите по ссылке <https://www.elastic.co/downloads/kibana#ga-release> и в зависимости от вашей операционной системы скачайте файл ZIP/TAR, как показано на рис. 7.1.



Сообщество разработчиков Elastic довольно активно, и релизы новых версий с обновленным функционалом и исправлениями выходят довольно часто. За время, которое вы уделили чтению этой книги, последняя версия Kibana могла измениться. Инструкции в книге актуальны для версии Kibana 6.0.0. Вы можете перейти по ссылке [past releases](#) (прошлые релизы) и скачать версию 6.0.0. Информация, приведенная в книге, будет действительна для любого релиза версии 6.x.

Download Kibana

Want to upgrade? We'll give you a hand. [Migration Guide »](#)

Version: 6.0.0

Release date: November 14, 2017

Notes: [View release notes.](#)
[Not the version you're looking for? View past releases.](#)

Downloads: [WINDOWS sha](#) [MAC sha](#) [LINUX 64-BIT sha](#)
[RPM 64-BIT sha](#) [DEB 64-BIT sha](#)

Рис. 7.1

Kibana — это инструмент визуализации, который опирается на Elasticsearch при выполнении запросов данных, используемых для создания визуализаций. Следовательно, прежде, чем продолжать, убедитесь, что у вас установлена и работает Elasticsearch.

Установка в Windows

Распакуйте скачанный файл. После распаковки перейдите в созданную папку, как показано в следующем фрагменте кода:

```
D:\>cd D:\packt\kibana-6.0.0-windows-x86_64
```

Для запуска Kibana перейдите в папку `bin`, наберите `kibana.bat` и нажмите клавишу **Enter**.

Установка в Linux

Распакуйте пакет `tar.gz` и перейдите в созданный каталог, как показано ниже:

```
$> tar -xzf kibana-6.0.0-darwin-x86_64.tar.gz  
$> cd kibana/
```

Для запуска Kibana перейдите в папку `bin`, наберите `./kibana` и нажмите клавишу **Enter**.

Вы должны получить логи следующего вида:

```
    log  [04:52:06.749]  [info][optimize]  
Optimizing and caching bundles for kibana,  
stateSessionStorageRedirect, timelion and  
status_page. This may take a few minutes  
    log  [04:55:20.118]  [info][optimize]  
Optimization of bundles for kibana,  
stateSessionStorageRedirect, timelion and  
status_page complete in 193.36 seconds  
    log  [04:55:20.241]  [info][status]  
[plugin:kibana@6.0.0] Status changed from  
uninitialized to green - Ready  
    log  [04:55:20.402]  [info][status]  
[plugin:elasticsearch@6.0.0] Status changed
```

```
from uninitialized to yellow - Waiting for
Elasticsearch
    log [04:55:20.426] [info][status]
[plugin:console@6.0.0] Status changed from
uninitialized to green - Ready
    log [04:55:20.454] [info][status]
[plugin:metrics@6.0.0] Status changed from
uninitialized to green - Ready
    log [04:55:21.987] [info][status]
[plugin:timelion@6.0.0] Status changed from
uninitialized to green - Ready
    log [04:55:22.001] [info][listening] Server
running at http://localhost:5601
    log [04:55:22.008] [info][status][ui
settings] Status changed from uninitialized to
yellow - Elasticsearch plugin is yellow
    log [04:55:22.270] [info][status]
[plugin:elasticsearch@6.0.0] Status changed
from yellow to green - Kibana index ready
    log [04:55:22.273] [info][status][ui
settings] Status changed from yellow to green -
Ready
```

Kibana является веб-приложением и работает на платформе Node.js, в отличие от Elasticsearch и Logstash, которые работают на JVM. Во время запуска Kibana пытается подключиться к Elasticsearch по адресу <http://localhost:9200>. Kibana запускается с портом 5601 по умолчанию. Вы можете использовать браузер для входа в Kibana по ссылке <http://localhost:5601>, а также увидеть статус сервера по ссылке <http://localhost:5601/status>.

Страница статуса показывает информацию об использовании ресурсов сервера и перечень установленных плагинов (рис. 7.2).

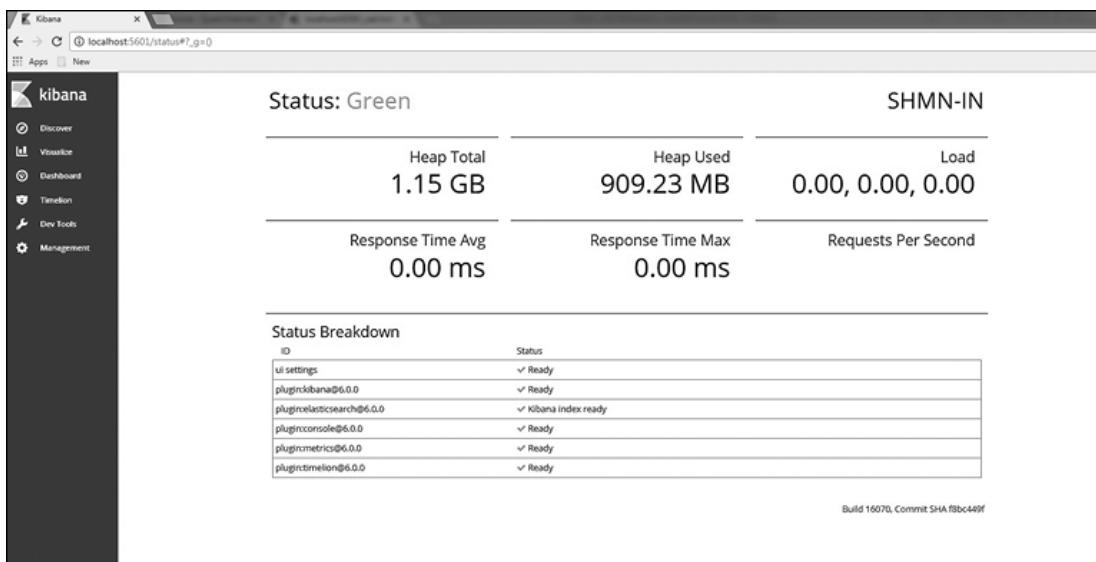


Рис. 7.2



Следует настраивать Kibana таким образом, чтобы она работала с узлом Elasticsearch той же версии. Совместная работа разных патч-версий Kibana и Elasticsearch (например, Kibana 6.0.0 и Elasticsearch 6.0.1) в целом поддерживается, но не рекомендуется.

Запуск разных крупных версий (например, Kibana 5.x и Elasticsearch 2.x) не поддерживается, равно как и запуск мелких версий Kibana, которые являются новее версии Elasticsearch (например, Kibana 6.1 и Elasticsearch 6.0).

Настройка Kibana

После запуска Kibana работает, используя порт 5601, и пытается подключиться к Elasticsearch на порте 9200. Что, если мы захотим изменить эти настройки? Вся конфигурация Kibana

находится в файле с названием `kibana.yml`, который расположен в папке `config` внутри папки `$KIBANA_HOME`. Если открыть этот файл в текстовом редакторе, можно увидеть, что его содержимое представляет собой большое количество настроек (пары «ключ — значение»), которые рекомендованы по умолчанию. Это значит, что, пока значение не было изменено, оно считается значением по умолчанию. Чтобы убрать комментарий с параметра, удалите символ `#` в начале строки параметра и сохраните файл.

В табл. 7.1 приведены ключевые настройки, которые пригодятся вам в начале работы с Kibana.

Таблица 7.1

<code>server.port</code>	Эта настройка указывает порт, с помощью которого Kibana будет выполнять запросы. Значение по умолчанию — 5601
<code>server.host</code>	Определяет адрес, с которым Kibana будет связываться. Корректными значениями являются и IP-адреса, и имена хостов. Значение по умолчанию — <code>localhost</code>
<code>elasticsearch.url</code>	URL-ссылка, используемая для всех запросов Elasticsearch. Значение по умолчанию — <code>http://localhost:9200</code> . Если вы настроили работу Elasticsearch на другой хост/порт, убедитесь, что изменили этот параметр соответствующим образом
<code>elasticsearch.username</code> <code>elasticsearch.password</code>	Если настроена авторизация в Elasticsearch, укажите в этом параметре имя пользователя и пароль для доступа. Более детально о безопасности Elasticsearch мы поговорим в следующей главе (X-Pack)
<code>server.name</code>	Легко читаемое имя отображения для идентификации процесса Kibana. Значение по умолчанию соответствует названию хоста
<code>kibana.index</code>	Kibana использует индекс в Elasticsearch для хранения выполненных поисков, визуализаций и панелей управления. Если индекс не существует, Kibana создаст его. Значение по умолчанию — <code>.kibana</code>



Файл .yml чувствителен к пробелам и отступам. Убедитесь, что все некомментированные параметры имеют равный отступ, в противном случае во время запуска Kibana произойдет ошибка, делающая невозможной дальнейшую работу.

Подготовка данных

Поскольку вся суть работы Kibana состоит в детальном рассмотрении данных, загрузим образец данных, которые мы будем использовать в процессе обучения. Один из самых распространенных примеров использования — анализ логов. Для этого примера мы загрузим логи сервера Apache в Elasticsearch через Logstash и далее будем использовать Kibana для анализа и создания визуализаций.

По ссылке <https://github.com/elastic/elk-index-size-tests> находится набор логов сервера Apache, которые были собраны с сайта www.logstash.net за период с мая по июнь 2014 года. Он содержит 300 000 лог-событий.

Перейдите по ссылке <https://github.com/elastic/elk-index-size-tests/blob/master/logs.gz> и нажмите кнопку **Download** (Скачать). Распакуйте файл logs.gz.

Убедитесь, что у вас установлена версия Logstash 5.6 или выше. Создайте в папке \$LOGSTASH_HOME\bin конфигурационный файл apache.conf, как показано во фрагменте кода ниже:

```
input
{
  file {
    path => "D:\Learnings\data\logs\logs"
    type => "logs"
```

```
        start_position => "beginning"
    }
}

filter
{
    grok {
        match => {
            "message" => "%{COMBINEDAPACHELOG}"
        }
    }
    mutate {
        convert => { "bytes" => "integer" }
    }
    date {
        match      =>      [      "timestamp",
"dd/MMM/YYYY:HH:mm:ss Z" ]
        locale => en
        remove_field => "timestamp"
    }
    geoip {
        source => "clientip"
    }
    useragent {
        source => "agent"
        target => "useragent"
    }
}

output
{
```

```
stdout {  
    codec => dots  
}  
elasticsearch { }  
}
```

Запустите Logstash, как показано ниже, чтобы программа приступила к обработке логов и их индексированию в Elasticsearch. Для запуска Logstash потребуется некоторое время, и в результате вы увидите серию точек (по точке на каждую обработанную строку логов):

```
$LOGSTASH_HOME\bin>logstash -f apache.conf
```

Проверим общее количество документов (лог-событий), индексированных в Elasticsearch:

```
curl -X GET http://localhost:9200/logstash-*/_count
```

В качестве ответа вы должны увидеть значение 300000.

Пользовательский интерфейс Kibana

Откройте Kibana в браузере по ссылке <http://localhost:5601>. Стартовая страница будет выглядеть следующим образом (рис. 7.3).

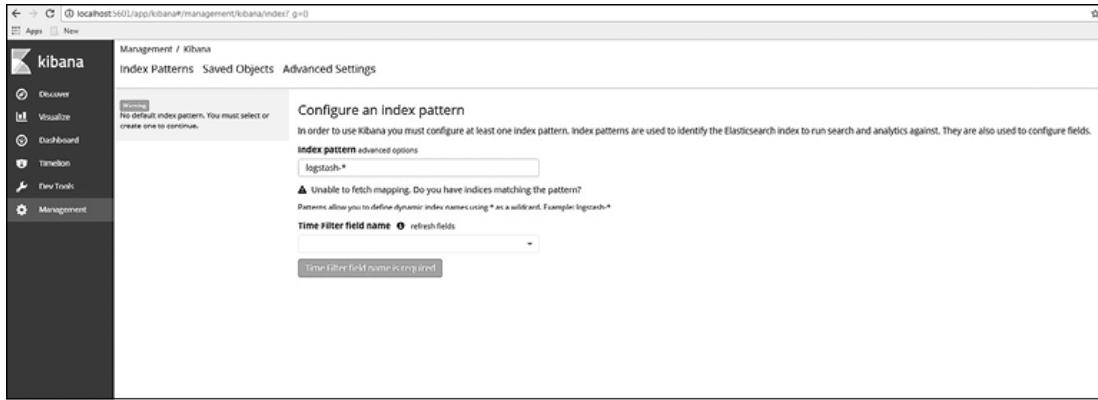


Рис. 7.3

Взаимодействие с пользователем

Прежде чем углубляться в ключевые компоненты Kibana, разберемся во взаимодействии с пользователем. Типичный процесс пользовательского взаимодействия изображен на рис. 7.4.



Рис. 7.4

По следующим пунктам вы сможете четко проследить процесс взаимодействия в Kibana.

1. До того как приступить к анализу данных в Kibana, пользователь загрузил данные в Elasticsearch.
2. Чтобы анализировать данные в Kibana, пользователю необходимо предоставить программе сведения о данных, которые хранятся в индексах ES. Он выбирает индексы, по которым необходимо провести анализ.
3. После настройки пользователю следует узнать структуру данных: поля документа, типы полей. Это требуется, чтобы принять решение о том, как именно необходимо

визуализировать данные, какие ставить вопросы и какие нужны ответы.

4. Далее, когда есть четкое понимание имеющихся данных, вопросов и желаемых ответов, пользователь создает необходимые визуализации, которые облегчат поиск ответов в огромном объеме данных.
5. Пользователь создает панель управления из набора визуализаций, сформированных ранее.
6. Этот процесс может повторяться на различных стадиях, пока пользователь не получит ответы на все свои вопросы касательно данных. В процессе он может увидеть более детальную картину по данным и сразу получить ответы на только что сформулированные вопросы, которые, возможно, изначально и не были поставлены.

Теперь, когда мы понимаем, как пользователь будет работать в Kibana, рассмотрим, из чего же состоит этот компонент. Как видно в левой части окна на рис. 7.3, пользовательский интерфейс Kibana состоит из следующих компонентов.

- **Discover (Исследование).** Эта страница помогает в исследовании данных, имеющихся в индексах ES. Она позволяет запрашивать данные, фильтровать их и анализировать структуру документов.
- **Visualize (Визуализация).** Страница предназначена для создания визуализаций. Вы можете найти различные варианты оформления: гистограммы, линейные диаграммы,

карты, облака тегов и пр. Пользователь может подобрать необходимые визуализации для облегчения анализа данных.

- **Dashboard (Панель управления).** Здесь вы можете собрать несколько визуализаций на единую страницу.
- **Timelion (Компонент Timelion).** На этой странице можно визуализировать данные в виде временного ряда с помощью простого языка выражений. Кроме того, пользователь может комбинировать независимые источники данных (из разных индексов) в одной визуализации.
- **Dev Tools (Инструменты разработчика).** Содержит набор плагинов, каждый из которых имеет определенный функционал. По умолчанию эта страница включает только один плагин — **Console** (Консоль), который предоставляет пользовательский интерфейс для взаимодействия с REST API в Elasticsearch.
- **Management (Управление).** На этой странице выполняется настройка индексов и управление ими. Здесь также можно управлять уже существующими визуализациями, панелями управления, поисковыми запросами (удалять, экспортить, импортировать).

Настройка шаблона индекса

Прежде чем вы сможете работать с данными и создавать визуализации для их анализа, требуется настроить шаблон индекса в Kibana. Такие шаблоны используются для идентификации индексов Elasticsearch, которые будут применяться для поиска и аналитики. Они также предназначены для настройки полей. Шаблон индекса представляет собой строку с возможными подстановочными

символами, которые могут соответствовать различным индексам. Обычно в Elasticsearch существует два типа индексов.

- **Временной индекс.** Если между данными и меткой времени есть взаимосвязь, то такие данные называются *данными временного ряда*. Для них характерно наличие поля с меткой времени. Примером таких данных могут служить логи, метрики, данные твитов. В Elasticsearch эти сведения хранятся в различных индексах, названия которых обычно присваиваются по метке времени, например `unixlogs-2017.10.10`, `tweets-2017.05`, `logstash-2017.08.10`.
- **Постоянные индексы.** Если данные не имеют меток времени или не связаны с временем, они называются *постоянными*. Обычно такие данные хранятся в одном индексе, например данные отдела или каталога продуктов.

Рассмотрим экран **Configure an Index Pattern** (Настройки шаблона индекса). Если в индексе есть поле даты/времени (то есть это временной индекс), то будет виден пункт **Time Filter field name** (Название поля фильтра времени). Этот пункт позволяет выбрать корректное поле даты/времени. В ином случае поле невидимо.

Поскольку в предыдущем разделе мы уже загрузили данные, настроим их, чтобы иметь возможность использовать во всех примерах этой главы. В поле **Index Name or Pattern** (Название или шаблон индекса) введите `logstash-*`. Для **Time Filter field name** (Название поля фильтра) введите `@timestamp` и нажмите кнопку **Create** (Создать).

Вы должны увидеть следующую страницу (рис. 7.5).

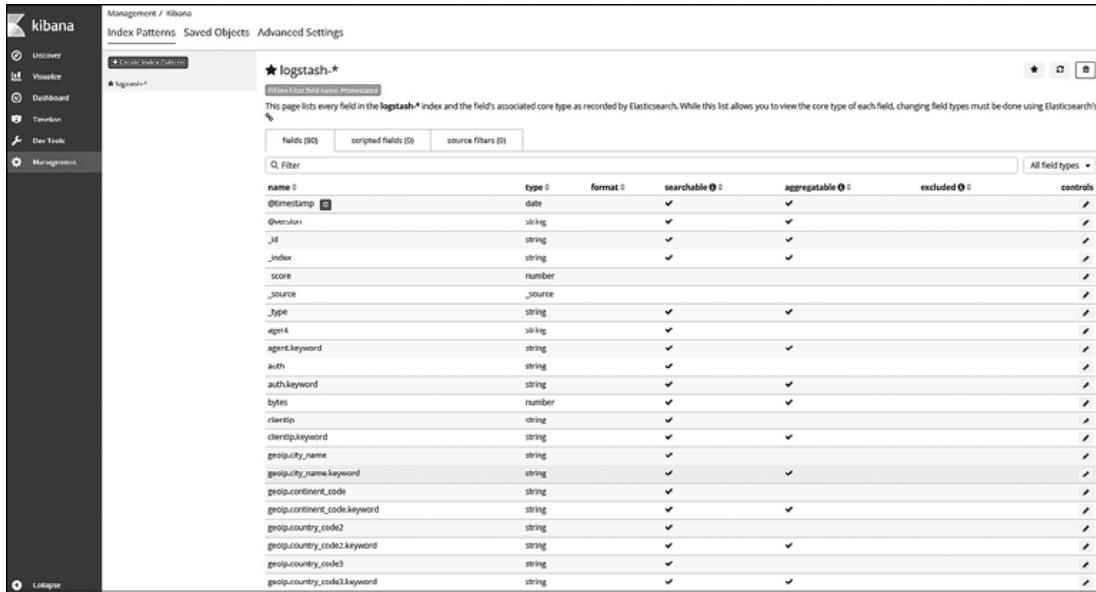


Рис. 7.5

Исследование

Страница **Discover** (Исследование) предназначена для интерактивного анализа данных. Вы можете выполнять запросы поиска, фильтрации, просматривать данные документов. Можно также сохранять поиск или критерии фильтрации для повторного использования или создания визуализаций на базе отфильтрованных результатов.

По умолчанию страница **Discover** (Исследование) показывает события за последние 15 мин. Укажите корректный диапазон даты в фильтре времени, так как мы загрузили данные за период с мая по июнь 2014 года. Перейдите на вкладку **Time Filter**—>**Absolute Time Range** (Фильтр времени—>Абсолютный диапазон времени) и установите следующие значения: в поле **From** (От) — 2014-05-28 00:00:00.000 и в поле **To** (До) — 2014-07-01 00:00:00.000 (рис. 7.6). Нажмите кнопку **Go** (Начать).



Рис. 7.6

Разделы страницы **Discover** (Исследование) вы можете увидеть на рис. 7.7.

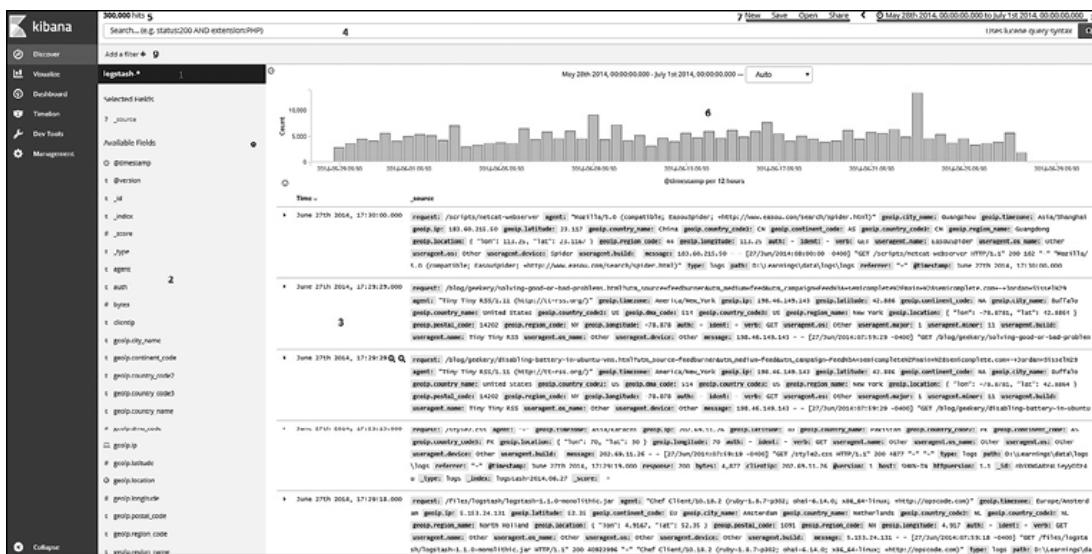


Рис. 7.7

Шаблон индекса (1), список полей (2), таблица документов (3), панель запросов (4), совпадения (5), гистограмма (6), панель инструментов (7), выбор времени (8) и фильтры (9).

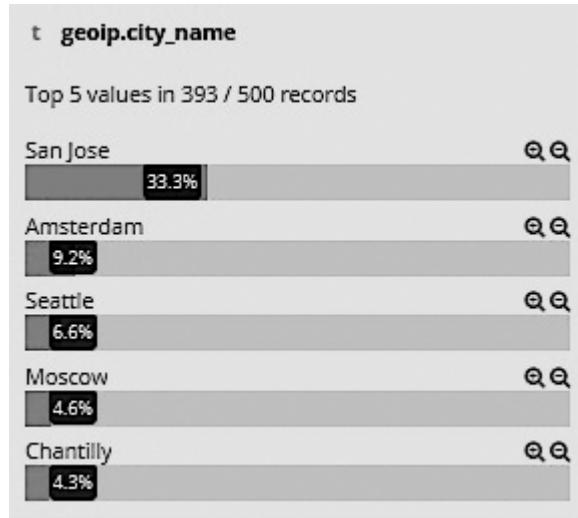


Рис. 7.8

Рассмотрим каждый из них.

- **Шаблон индекса.** Здесь вы можете найти все настроенные шаблоны индекса, шаблон по умолчанию выбирается автоматически. Пользователю разрешается выбрать подходящий шаблон индекса для исследования данных.
- **Список полей.** Показаны все поля, которые являются частью документа. Если щелкнуть на поле **Quick Count** (Быстрый подсчет), вы получите общее количество документов в таблице документов, которые содержат определенное поле, топ-5 значений, а также процент документов, содержащих каждое значение (рис. 7.8).
- **Таблица документов.** Показаны актуальные данные документов. Таблица показывает последние 500 документов, соответствующих пользовательскому запросу/фильтрам, с сортировкой по метке времени (если поле существует). При нажатии кнопки **Expand** (Расширить) слева вы получаете возможность визуализации данных в формате таблицы или в формате JSON (рис. 7.9).

The screenshot shows a search results table in the Logstash interface. At the top, there is a header row with columns for Time and _source. Below this is a table with rows for various log fields like @timestamp, _id, _index, _score, _type, agent, auth, bytes, clientip, geoip.city_name, geoip.continent_code, and geoip.country_code. The table has a 'Table' tab selected at the bottom left.

Рис. 7.9

В процессе исследования данных мы нередко больше заинтересованы в определенном наборе полей, чем в документе целиком. Чтобы добавить поля в таблицу документов, вам необходимо навести указатель мыши на поле или список полей, после чего нажать кнопку **Add** (Добавить). Или разверните документ и нажмите кнопку **Toggle column in table** (Настроить столбец в таблице) (рис. 7.10).



Рис. 7.10

Добавленное поле заменяет столбец `_source` в таблице документов. Вы можете изменять порядок для столбцов полей в таблице, нажимая стрелки влево или вправо, которые отображаются при наведении указателя мыши на название столбца. Аналогичным образом при нажатии

кнопки удаления **X** вы можете удалять столбцы из таблицы (рис. 7.11).

Time	geoip.city_name	response	request X «
» June 27th 2014, 17:43:20.000	Guangzhou	200	Move column to the left X «
» June 27th 2014, 17:42:49.000	Buffalo	200	/blog/geekery/solving-good-or-bad-problems.html? utm_source=feedburner&utm_medium=feed&utm_campaign=Fe
» June 27th 2014, 17:42:49.000	Buffalo	200	/blog/geekery/disabling-battery-in-ubuntu-vms.html? utm_source=feedburner&utm_medium=feed&utm_campaign=Fe
» June 27th 2014, 17:42:39.000	-	200	/style2.css
» June 27th 2014, 17:42:38.000	Amsterdam	200	/files/logstash/logstash-1.1.0-monolithic.jar
» June 27th 2014, 17:42:37.000	-	200	/images/jordan-80.png
» June 27th 2014, 17:42:35.000	-	200	/reset.css
» June 27th 2014, 17:42:30.000	-	200	/blog/tags/X11
» June 27th 2014, 17:42:12.000	-	200	/images/googledotcom.png

Рис. 7.11

- **Панель запросов.** Используя панель запросов/панель поиска, пользователь может вводить запросы для фильтрации результатов. Запрос результатов поиска приводит к обновлению гистограммы (если для текущего шаблона индекса настроено поле времени), а также к обновлению таблицы документов, списков полей и совпадений, чтобы они соответствовали результатам поиска. Соответствующий поиску текст будет выделен в таблице документов. Для поиска по вашим данным введите поисковый запрос в поле запроса и нажмите **Enter** или щелкните на значке поиска.

Панель запросов принимает два вида запросов.

- Строковый запрос в Elasticsearch, который создается на базе синтаксиса запроса Lucene: https://lucene.apache.org/core/2_9_4/queryparsersyntax.html.

- Полный запрос DSL JSON в Elasticsearch: <https://www.elastic.co/guide/en/elasticsearch/reference/5.5/query-dsl.html>.

Детально рассмотрим оба варианта.

Строковый запрос в Elasticsearch

Предоставляет возможность выполнять разные типы поиска: от простого до сложного запросов, придерживающихся синтаксиса запроса Lucene. Рассмотрим несколько примеров.

Общий текстовый поиск. Для поиска текста по любому из полей попросту введите текстовую строку в поле запроса (рис. 7.12).



Рис. 7.12

Когда вы вводите несколько слов для поиска, в результаты включаются все документы, которые содержат хотя бы одно слово из искомых, или все слова в любом порядке.

Если вы хотите найти конкретную фразу, то есть найденные документы должны содержать все указанные слова в указанном порядке, то введите фразу в кавычках, например "file logstash" или "files logstash".

Поиск по полю. Для поиска по значениям в определенных полях используйте поле `syntax: value` (рис. 7.13).

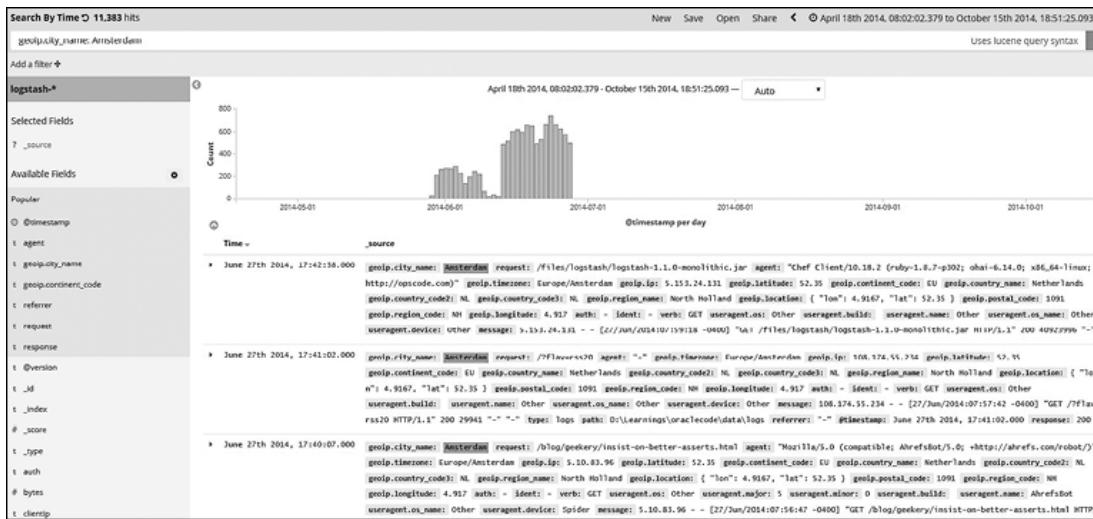


Рис. 7.13

Булев поиск. Возможно применять булевые операторы, такие как AND, OR и - (Must Not Match) для создания сложных запросов. Используя булевые операторы, можно комбинировать поля: `value` и обычный текст (рис. 7.14).



Имейте в виду, что операторы AND и OR чувствительны к регистру.



Рис. 7.14

Соответствие Must Not. Далее вы можете увидеть пример применения оператора Must Not с полем (рис. 7.15).

Далее вы можете увидеть пример использования оператора Must Not с обычным текстом (рис. 7.16).



Между оператором - и поисковым текстом/полем не должно быть пробела.

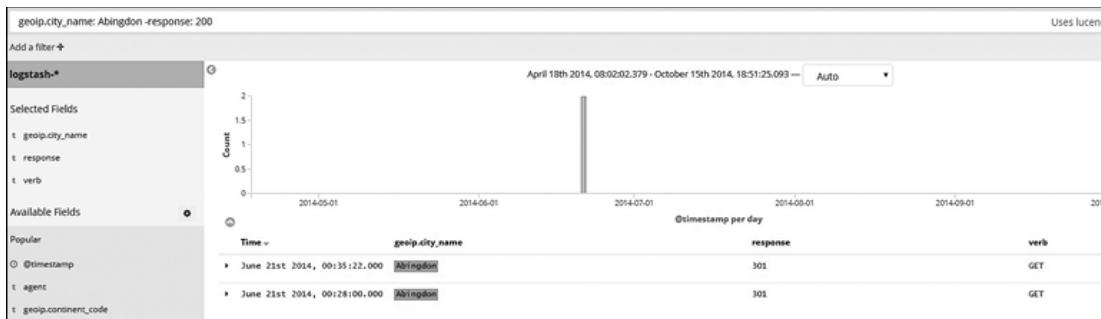


Рис. 7.15



Рис. 7.16

Групповой поиск. Когда мы хотим создать сложный запрос, зачастую приходится группировать критерии поиска. Поддерживается группирование и по полю, и по значению, как можно увидеть на примере ниже (рис. 7.17).



Рис. 7.17

Поиск диапазона. Позволяет искать среди диапазона значений. Включающие диапазоны указываются в квадратных скобках, например [START_VALUE TO END_VALUE], а исключающие диапазоны — в фигурных, например { START _VALUE TO END_VALUE }. Вы можете указать диапазоны для дат, числовых или строковых полей (рис. 7.18).

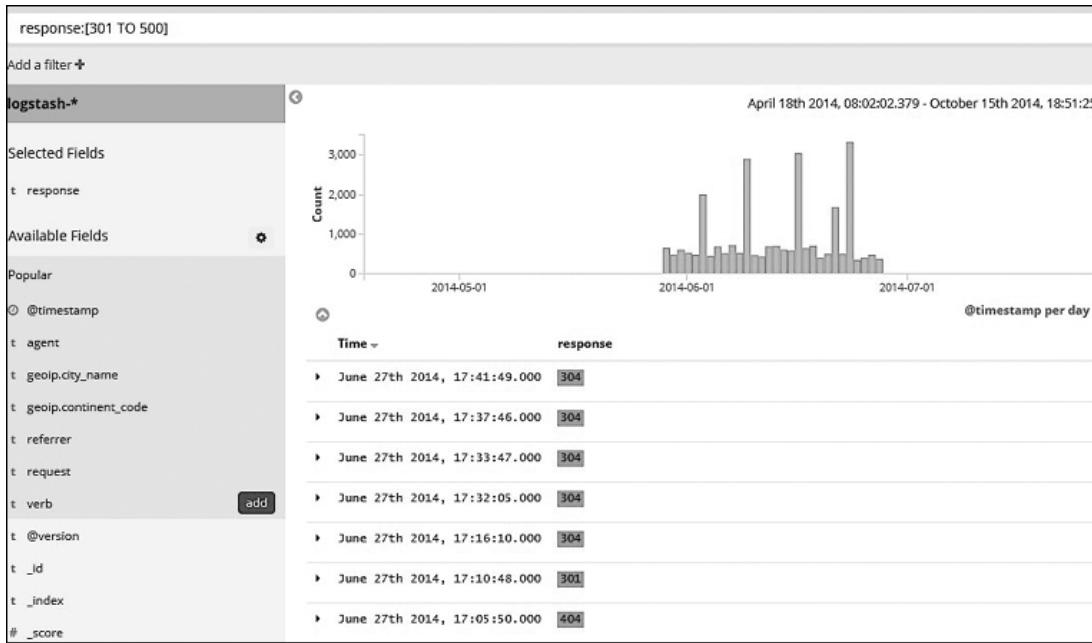


Рис. 7.18



Оператор ТО чувствителен к регистру; его значения диапазона должны быть числовыми.

Поиск подстановочного символа и регулярного выражения. Вы можете выполнять запросы, используя символы * и ? с поисковым текстом. Символ * определяет «ни одного соответствия или несколько», а ? задает «ни одного соответствия или одно» (рис. 7.19).



Поиск с подстановочным символом может быть требователен к ресурсам. Всегда рекомендуется добавлять подстановочный символ как суффикс, а не как префикс к поисковому тексту.

По аналогии поддерживаются и regex-запросы. Вы можете указывать шаблоны regex, используя слеш (/) и квадратные скобки ([]). Имейте в виду, что regex-запросы также крайне требовательны к вычислительной мощности.

Например, найдем любые города, которые начинаются с букв g, b или a (рис. 7.20).

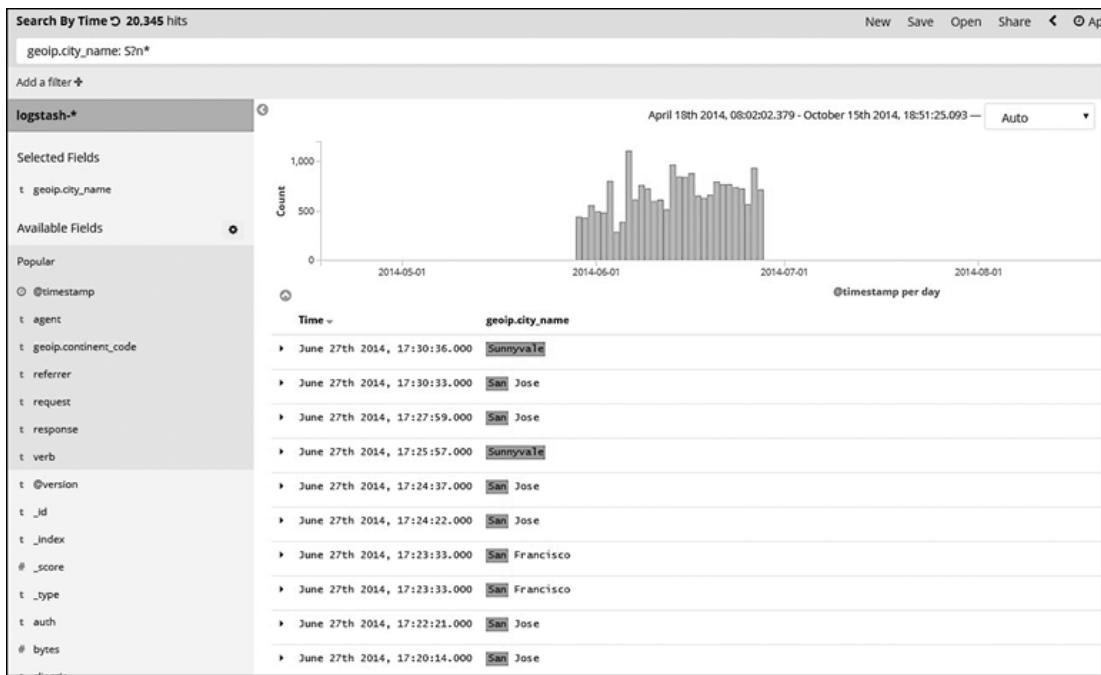


Рис. 7.19

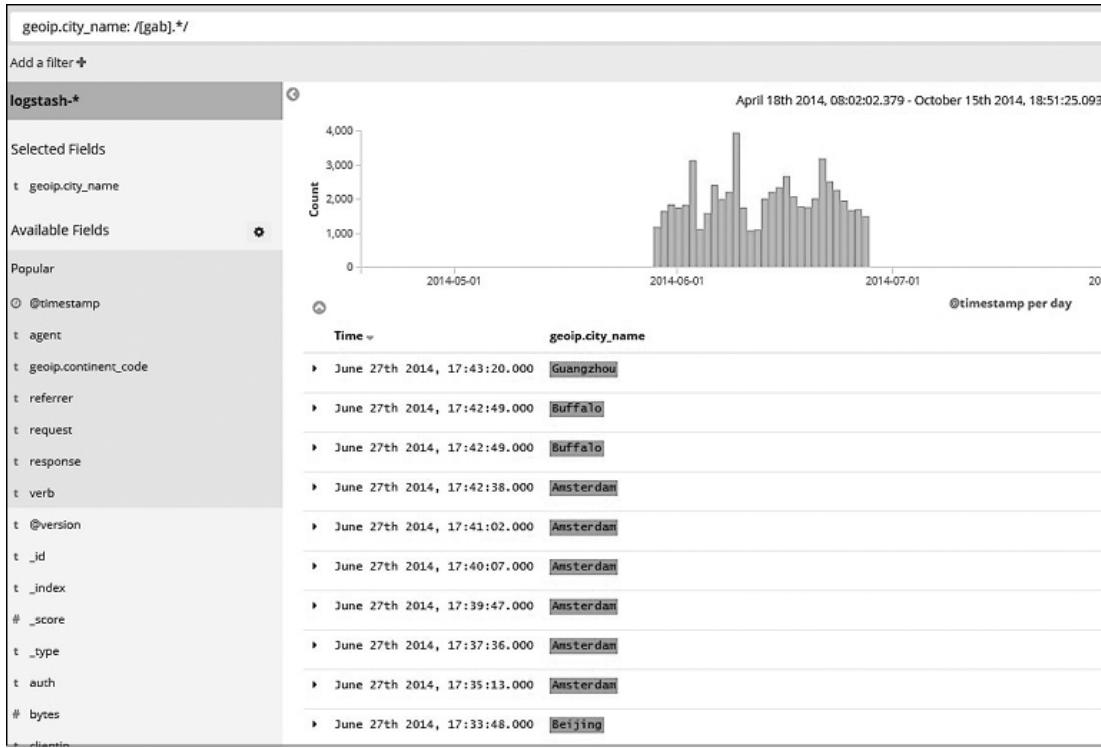


Рис. 7.20

Запрос DSL в Elasticsearch

С помощью запроса DSL можно выполнять запросы напрямую с панели запросов. Его также можно применять для поиска.

На рис. 7.21 вы можете увидеть пример поиска документов, которые имеют значение IE в useragent.name и значение Washington в поле geoip.region_name.

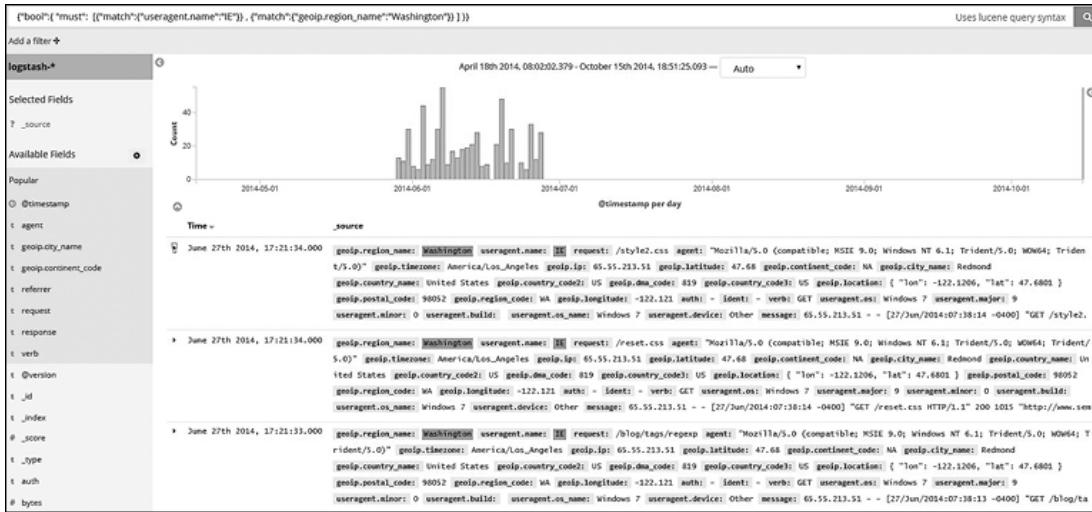


Рис. 7.21

Совпадения. Обозначает общее количество документов, которые соответствуют запросу/критерию, введенному пользователем.

Гистограмма. Этот раздел доступен только при настроенном поле времени для выбранного шаблона индекса. Вы можете увидеть распределение документов по времени с помощью гистограммы. По умолчанию лучший интервал времени для создания гистограммы рассчитывается на основе данных, установленных в фильтре времени. Но вы можете определить другое значение интервала в соответствующем раскрывающемся меню (рис. 7.22).

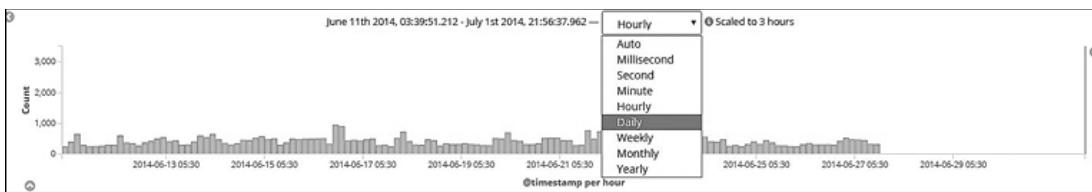


Рис. 7.22

Во время исследования данных пользователь может «разрезать» гистограмму на секторы и фильтровать результаты поиска. При наведении указателя мыши на гистограмму он

меняет изображение на +. Нажав и удерживая кнопку мыши, пользователь может нарисовать прямоугольную область для исследования/фильтрации документов, которые попадают в выбранные интервалы.



После «нарезки» гистограммы интервал/период времени меняется. Для возврата нажмите в браузере кнопку Back (Назад).

Панель инструментов. Введенные пользователем поисковые запросы и примененные фильтры можно сохранить для повторного использования в дальнейшем или для создания визуализаций на базе отфильтрованных результатов поиска. Панель инструментов предоставляет такие функции, как сохранение, повторное использование и распределение поисковых запросов. Пользователь может вернуться к сохраненному поисковому запросу и изменить его, после чего перезаписать существующий поиск или сохранить результат в виде нового поиска (установив флажок **Save as new search** (Сохранить как новый поиск) в окне **Save** (Сохранить)).

Создадим новый поиск, базируясь на существующем сохраненном поиске (рис. 7.23).

The screenshot shows a search results page titled "Response_200" with 18 hits. At the top right, there are buttons for "New", "Save", "Open", "Share", and a timestamp "June 18th 2014, 11:58:23.859". Below the title, there is a "Save Search" button. The main area displays a search result for "Response_200_Graz". A checkbox labeled "Save as a new search" is checked, and a "Save" button is located below it. The bottom of the screen shows the search query "response:200 AND geolp.city_name:Graz".

Рис. 7.23

После нажатия кнопки **Open** (Открыть) мы видим **Saved Searches** (Сохраненные поиски) (рис. 7.24).

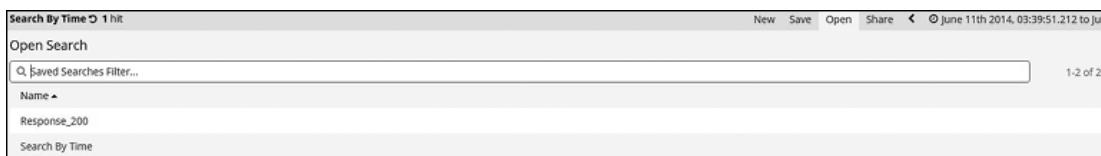


Рис. 7.24

В Kibana состояние текущей страницы/пользовательского интерфейса сохраняется в самой ссылке URL — так его проще делиться. Нажатие кнопки **Share** (Поделиться) позволяет вам поделиться сохраненным поиском (рис. 7.25).

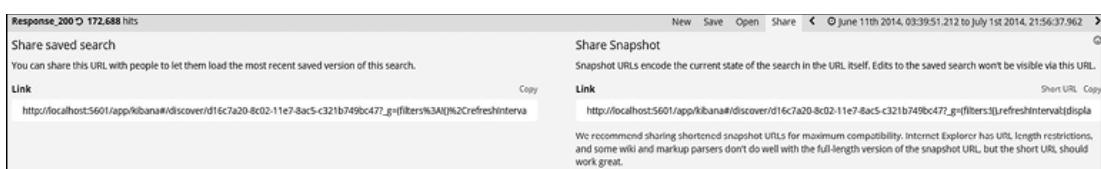


Рис. 7.25

Выбор времени. Этот раздел доступен только при настроенном поле времени для выбранного шаблона индекса. Фильтр времени ограничивает результаты поиска по определенному периоду времени, тем самым помогая анализировать данные, которые принадлежат к интересующему вас периоду времени. Если открыта страница **Discover** (Исследование), фильтр времени по умолчанию настроен на последние 15 мин.

Для выбора периодов времени фильтр времени предоставляет следующие параметры (отображаются при щелчке на **Time Filter** (Фильтр времени)).

- **Быстрый фильтр времени.** Задает быструю фильтрацию с использованием предустановленных диапазонов времени (рис. 7.26).

Time Range		Quick	Relative	Absolute
Today	Yesterday	Last 15 minutes	Last 30 days	
This week	Day before	Last 30 minutes	Last 60 days	
This month	yesterday	Last 1 hour	Last 90 days	
This year	This day last week	Last 4 hours	Last 6 months	
The day so far	Previous week	Last 12 hours	Last 1 year	
Week to date	Previous month	Last 24 hours	Last 2 years	
Month to date	Previous year	Last 7 days	Last 5 years	
Year to date				

Рис. 7.26

- **Относительный фильтр времени.** Помогает выполнять фильтрацию на базе относительного времени, то есть относительно текущего времени. Относительное время может быть в прошлом или будущем. Для округления времени предусмотрен соответствующий флажок (рис. 7.27).

Time Range		Quick	Relative	Absolute
From	Set To Now	To	Set To Now	
December 15th 2013, 12:28:47.432		December 15th 2014, 12:28:47.432		
<input type="text" value="4"/>	<input type="button" value="Years ago"/> ▾	<input type="text" value="3"/>	<input type="button" value="Years ago"/> ▾	
<input type="checkbox"/> round to the year		<input type="checkbox"/> round to the year		
Go				

Рис. 7.27

- **Абсолютный фильтр времени.** Позволяет вам фильтровать данные с ориентацией на введенное начальное и конечное время (рис. 7.28).

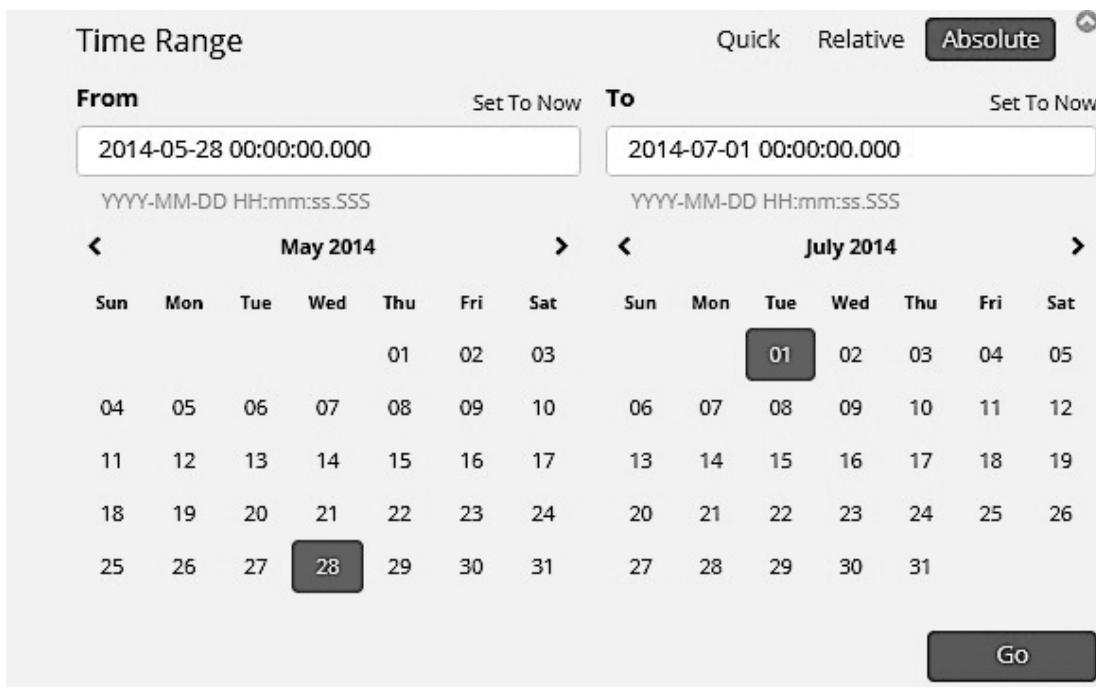


Рис. 7.28

- **Автоматическое обновление.** Во время анализа в реальном режиме времени данные генерируются постоянно. Потому пригодится возможность автоматически считывать актуальные данные. Именно для этого служит автоматическое обновление. По умолчанию интервал обновления не определен. Вы можете самостоятельно выбрать необходимый для вашего анализа интервал (рис. 7.29).

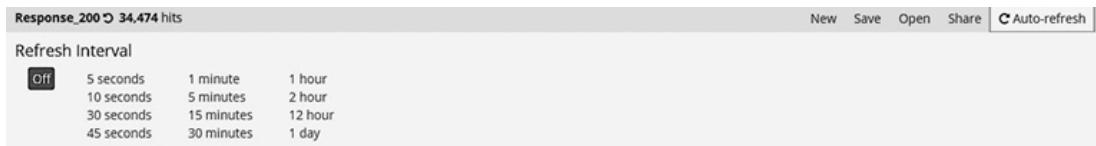


Рис. 7.29



Фильтр времени доступен на страницах Discover (Исследование), Visualize (Визуализация) и Dashboard (Панель управления). Диапазон времени, который выбирается/устанавливается на этих страницах, может быть перенесен и на другие страницы.

Фильтр времени есть даже на странице Timelion (Компонент Timelion), однако он не зависит от настроек времени на других страницах.

- **Фильтры.** Используя положительные (positive) фильтры, можно оптимизировать результаты поиска так, чтобы отображались только те документы, которые содержат в поле определенное значение. Допускается также создавать отрицательные (negative) фильтры, которые будут исключать документы, содержащие определенное значение поля.

Вы можете добавлять фильтры на страницах **Fields List** (Список полей) или **Documents Table** (Таблица документов), в том числе вручную. На странице **Documents Table** (Таблица документов) можно также фильтровать документы по наличию или отсутствию поля.

Для добавления положительного или отрицательного фильтра перейдите на страницу **Fields List** (Список полей) или **Documents Table** (Таблица документов) и щелкните на значке + или - соответственно. Аналогичным образом для фильтрации по наличию или отсутствию поля щелкните на значке * (фильтр наличия) (рис. 7.30).

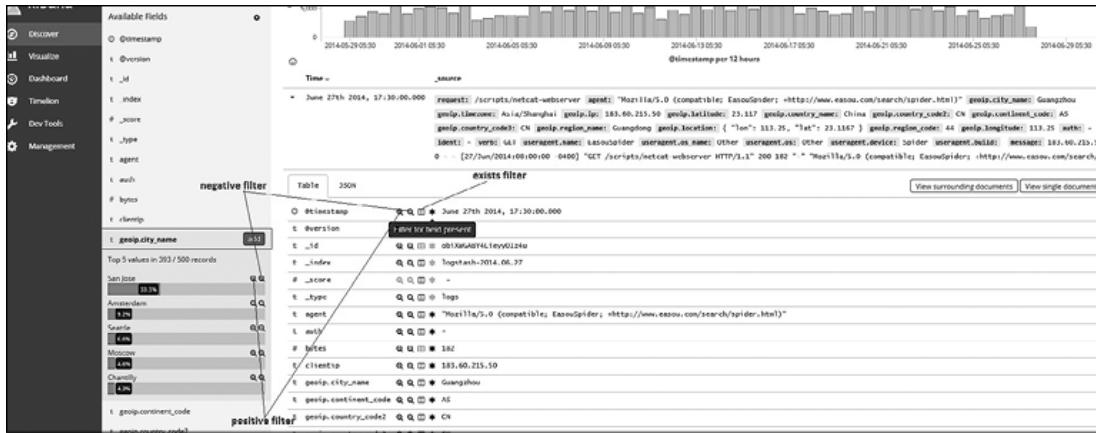


Рис. 7.30

Для добавления фильтров вручную нажмите кнопку **Add a Filter** (Добавить фильтр), которая расположена ниже на панели запроса. Это приведет к появлению всплывающего окна, в котором вы можете выбрать и применить фильтры, нажав кнопку **Save** (Сохранить) (рис. 7.31).

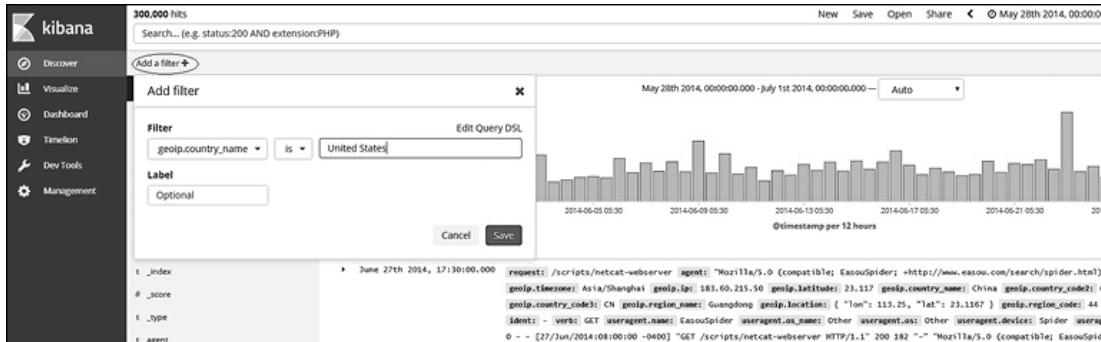


Рис. 7.31

Примененные фильтры показаны на панели запроса на рис. 7.32. Отрицательные фильтры будут выделены красным. Вы можете добавить несколько фильтров и применить к ним следующие действия.

- **Enable/Disable Filter (Включение/выключение фильтра).** Эта кнопка позволяет активизировать или

деактивизировать фильтр без его удаления. Диагональные линии указывают на то, что фильтр выключен.

- **Pin Filter (Закрепить фильтр).** Закрепляет фильтр. Такие фильтры сохраняются при изменении контекстов в Kibana. Например, вы можете закрепить фильтр на странице **Discovery** (Исследование), и он останется на том же месте, когда вы переключитесь на страницу **Visualize** (Визуализация) или **Dashboard** (Панель управления).
- **Toggle Filter (Настроить фильтр).** Позволяет переключаться между положительным и отрицательным фильтром и наоборот.
- **Remove Filter (Удалить фильтр).** Дает возможность удалить примененный фильтр.
- **Edit Filter (Редактировать фильтр).** Позволяет редактировать примененный фильтр.

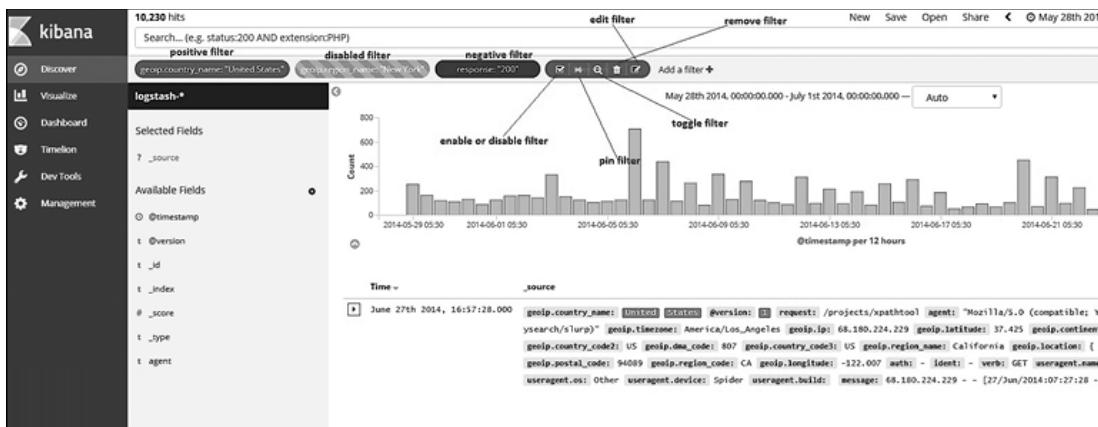


Рис. 7.32

Визуализация

Страница **Visualize** (Визуализация) служит для создания визуализаций в виде графиков, таблиц и диаграмм, благодаря чему можно легко визуализировать все данные, которые хранятся в Elasticsearch. Создавая визуализации, пользователь легко может увидеть свои данные со стороны, а также получить ответы на вопросы, которые могут возникнуть в процессе исследования. Уже созданные визуализации допускается сохранять и применять для создания панелей управления.

В случае с логами Apache пользователь сможет легко найти ответы на некоторые типичные вопросы, связанные с анализом логов.

- Как различается трафик в разных регионах мира?
- Какие ссылки запрашиваются чаще всего?
- С каких IP-адресов чаще всего запрашиваются ссылки?
- Как различается нагрузка сети в разное время?
- Есть ли подозрительные или мошеннические действия с каких-либо IP-адресов или регионов?

Все визуализации в Kibana основаны на запросах агрегации в Elasticsearch. Агрегации предоставляют группирование многомерных результатов. Например, можно найти самые распространенные браузеры по устройствам и по странам. В Kibana представлен широкий ассортимент визуализаций (рис. 7.33).

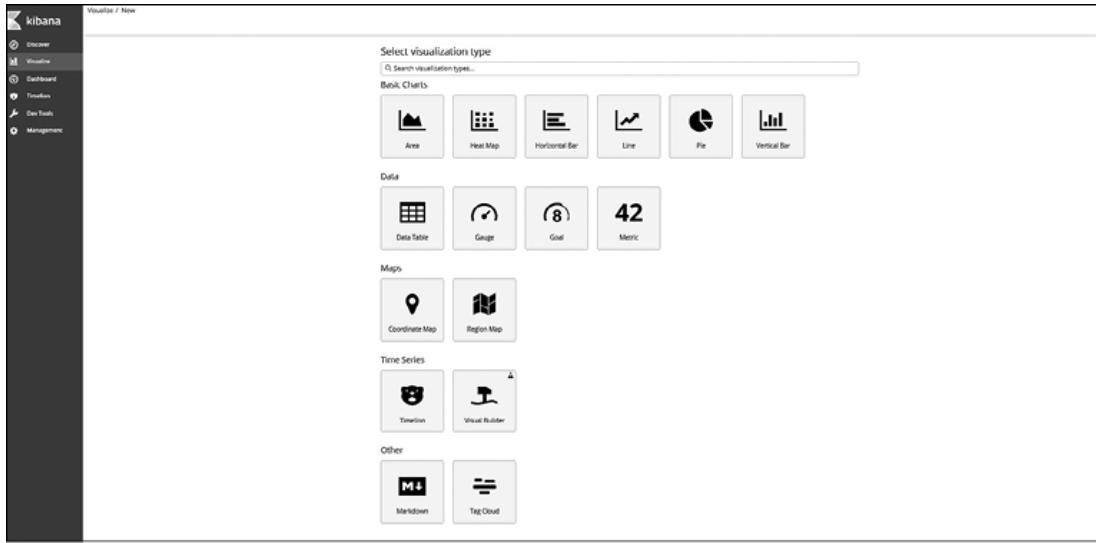


Рис. 7.33

Агрегации Kibana

Kibana поддерживает два типа агрегаций:

- сегментарные;
- метрические.

Поскольку для понимания принципа создания визуализаций важно разбираться в агрегациях, рассмотрим их до того, как приступать к визуализациям.

Сегментарные агрегации. *Сегментированием* называется группирование документов согласно общим критериям. Сегментирование работает аналогично запросу GROUP BY в SQL. В зависимости от типа агрегации каждый сегмент ассоциирован с критериями, которые определяют, соответствует ли данный документ выбранному сегменту. Для каждого сегмента доступна информация об общем количестве документов в нем.

Сегментарные агрегации могут применяться для следующих целей.

- Учет документов сотрудников в рамках соответствующего индекса.
- Поиск среди сотрудников в зависимости от их возраста или места работы.
- Поиск в индексах логов Apache вхождений ответов 404 в зависимости от страны.

Сегментарные агрегации поддерживают субагрегации, то есть при наличии сегмента все документы в нем могут и далее сегментироваться (группироваться по критериям). Пример: найти вхождения ответов 404 в зависимости от страны или штата.

В зависимости от типа сегментарной агрегации одни указывают единый сегмент, другие — фиксированное количество сегментов, третьи динамически создают сегменты в процессе агрегации.

Сегментарные агрегации могут быть совмещены с метрическими, например для поиска среднего возраста сотрудников в каждой возрастной группе.

Kibana поддерживает следующие типы сегментарных агрегаций.

- **Гистограмма.** Это тип агрегации гистограммы, который работает только с числовыми полями. Имея значение числового поля и интервал, агрегация разделяет их на сегменты фиксированного размера. Например, гистограмма может быть использована для поиска продуктов в определенном диапазоне цен с интервалом 100.

- **Гистограмма даты.** Это тип агрегации гистограммы, который работает только с полями даты. Принцип работы заключается в разделении их на сегменты фиксированного размера. Поддерживаются интервалы, основанные на дате или времени: два часа, два дня, две недели и т.д. Kibana предоставляет различные удобные интервалы, включая автоматический выбор, миллисекунды, секунды, минуты, часы, дни, недели, месяцы, годы и пользовательские настройки. Используя опцию **Custom** (Пользовательский), можно устанавливать ориентированные на дату/время интервалы. Данная гистограмма идеально подходит для анализа данных временного ряда, например, когда нужно найти общее количество входящих веб-запросов за неделю/день.
- **Диапазон.** Работает по аналогии с агрегациями гистограммы, однако вместо фиксированных интервалов указываются диапазоны. Возможна работа не только с числами, но и с датами и IP-адресами. Вы можете указывать несколько диапазонов, используя значения `from` и `to`. Например, агрегацию можно использовать для поиска количества сотрудников, которые попадают под возрастные диапазоны 0–25, 25–35, 35–50 и 50 лет и старше.
- **Условия.** Эта агрегация группирует документы, базируясь на каждом уникальном условии в поле. Она идеальна для поиска n значений, например поиска топ-5 стран на основе количества входящих веб-запросов.



Эта агрегация работает только с полями keyword.

- **Фильтры.** Агрегация используется для создания сегментов на основе условия фильтра. Она позволяет сравнивать определенные значения. Пример: поиск среднего количества веб-запросов в Индии по сравнению с США.
- **Сетка GeoHash.** Эта агрегация работает с полями, содержащими значения geo_point. В процессе работы поля geo_point группируются в сегменты и отображаются на карте. Пример: визуализация трафика веб-запросов по разным регионам.
- **Метрика.** Используется для вычисления метрик на основе значений, извлеченных из полей документов. Метрики применяются в связке с сегментами. Доступны следующие метрики.
 - **Подсчет.** Метрика по умолчанию в визуализациях Kibana, показывает количество документов.
 - **Среднее число.** Используется для подсчета среднего значения (для поля) для всех документов в сегменте.
 - **Сумма.** Применяется для подсчета суммы значений (для поля) для всех документов в сегменте.
 - **Медиана.** Предназначена для подсчета медианы значений (для поля) для всех документов в сегменте.
 - **Минимум.** Используется для подсчета минимального значения (для поля) для всех документов в сегменте.

- **Максимум.** Применяется для подсчета максимального значения (для поля) для всех документов в сегменте.
- **Стандартное отклонение.** Предназначена для подсчета стандартного отклонения значений (для поля) для всех документов в сегменте.
- **Процентиль.** Используется для подсчета процентиля значений (для поля) для всех документов в сегменте.
- **Ранг процента.** Используется в отношении набора процентилей для вычисления подобных значений.

Создание визуализации

Выполните следующие шаги для создания визуализаций.

1. Перейдите на страницу **Visualize** (Визуализировать) и нажмите кнопку **Create a new Visualization** (Создать новую визуализацию) или кнопку +.
2. Выберите тип визуализации.
3. Выберите источник данных.
4. Создайте визуализацию.

Интерфейс визуализации выглядит следующим образом (рис. 7.34).

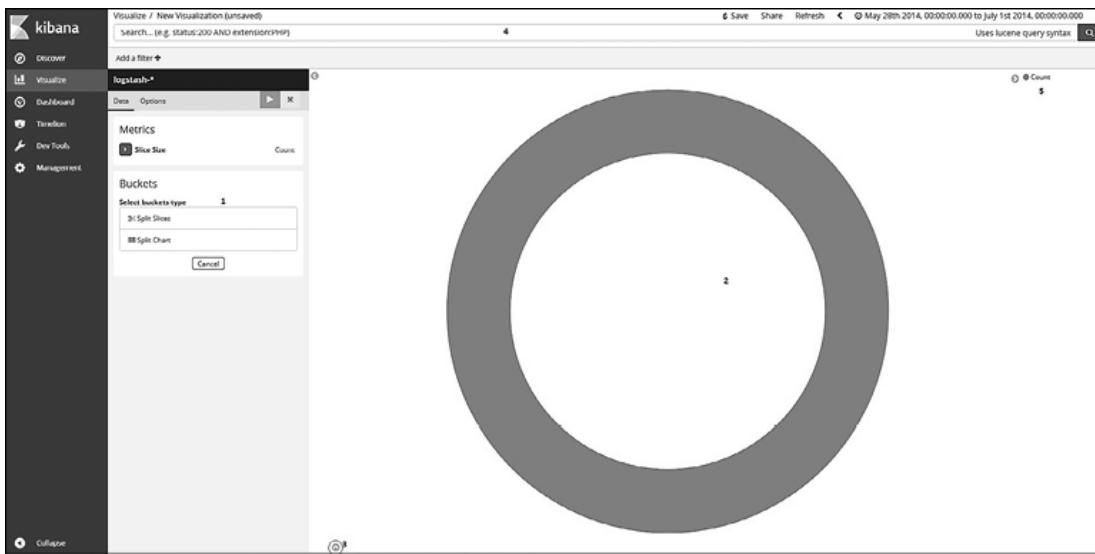


Рис. 7.34

Ниже перечислены компоненты интерфейса визуализации, которые показаны на рис. 7.34.

- **Дизайн визуализаций.** Используется для выбора подходящих метрик и сегментов для создания визуализаций.
- **Предпросмотр визуализаций.** В зависимости от выбранных метрик, сегментов, запросов, фильтров и времени визуализация динамически меняется.
- **Панель шпиона.** Позволяет исследовать необработанные запросы Elasticsearch, ответы, табличные данные и статистику запросов HTTP.
- **Панель запроса/Фильтры полей.** Применяется для фильтрации результатов поиска.
- **Метка.** Показывает тип метрики и ключи сегментов как метки. Цвета визуализации можно изменить; для этого нажмите кнопку **Label** (Метка) и выберите цвет на палитре.

- **Панель инструментов/Фильтр времени.** Предоставляет возможность сохранять визуализации и делиться ими. Кроме того, с помощью фильтра времени пользователь может ограничить время для фильтрации результатов поиска.



Фильтры времени, панель запросов и фильтры полей рассматривались в разделе «Исследование» ранее.

Типы визуализаций

Рассмотрим каждый тип визуализаций детально.

Линейные диаграммы, гистограммы, диаграммы областей

Эти диаграммы используются для визуализации распределенных данных путем сопоставления их осям X/Y. Их также можно задействовать для визуализации данных временного ряда и анализа полей. Гистограммы и диаграммы областей полезны для визуализации суммированных данных (когда используются субагрегации).

В версии Kibana 5.5 и новее предоставляется возможность динамически менять тип диаграммы. Иначе говоря, можно начать с диаграммы, но в процессе изменить тип на гистограмму или область. Тем самым достигается гибкость при выборе подходящего для анализа типа визуализации.

Таблица данных

Используется для отображения агрегированных данных в формате таблицы. Полезна для анализа данных с высоким уровнем расхождения, которые будет трудно анализировать с помощью диаграмм. Например, таблицу данных удобно применять для нахождения топ-20 ссылок или топ-20 IP-адресов. Кроме того, она помогает идентифицировать топ-*n* агрегаций.

Виджет MarkDown

Эта визуализация используется для создания форматированного текста, содержащего общую информацию, комментарии и инструкции, имеющие отношение к панели управления. Данный виджет принимает текст GitHub Markdown (более детально описан по ссылке <https://help.github.com/categories/writing-on-github/>).

Метрика

Метрические агрегации работают только с числовыми полями и отображают единое числовое значение для выбранных агрегаций.

Цель

Цель — метрическая агрегация, предоставляющая визуализации, демонстрирующие, как метрика достигает указанной цели. Это новый тип визуализации, представленный в Kibana 5.5.

Шкала

Шкала — метрическая агрегация, предоставляющая визуализации для отображения того, как метрические значения относятся к предустановленному порогу/диапазону. Например, такая визуализация может использоваться для демонстрации того, находится ли нагрузка на сервер в допустимом диапазоне, или же она достигла критического значения. Это новый тип визуализации, представленный в Kibana 5.5.

Круговые диаграммы

Эта визуализация применяется для отображения части относительно целого. В визуализации части показаны как сегменты.

Карты координат

Визуализация предназначена для отображения географических данных на карте в зависимости от сегментов/агрегаций. Чтобы можно было использовать эту агрегацию, документы должны иметь поля типа `geo_point`. Будет использована агрегация сетки GeoHash, точки будут сгруппированы в сегменты, которые представляют клетки на карте. Ранее эта визуализация называлась плиточной картой (`tile map`).

Карты регионов

Карты регионов — это тематические карты, в которых векторы границ подсвечены градиентом; более интенсивные цвета показывают высокие значения, а менее интенсивные — низкие. Известны под названием *картограмм* или *хороплетов* (https://ru.wikipedia.org/wiki/Фоновая_картограмма). По умолчанию Kibana предлагает два векторных слоя: один для

стран мира и один — для США. Это новый тип визуализации, представленный в Kibana 5.5.

Облако тегов

Облако тегов — визуальное отображение текстовых данных, которое обычно используется для визуализации текста свободной формы. Теги — это чаще всего единичные слова; важность каждого слова определяется размером или цветом шрифта. Размер шрифта задается метрической агрегацией. Например, если применяется метрика подсчета, то самое популярное слово имеет наибольший размер шрифта, а самое редкое — наименьший.

Визуализации в деле

Рассмотрим, как различные визуализации могут помочь решить следующие задачи.

- Анализ кодов ответов по времени.
- Поиск топ-10 запрошенных ссылок.
- Анализ использования сетевого трафика для каждой из топ-5 стран по времени.
- Поиск наиболее распространенного пользовательского агента.
- Анализ веб-трафика из различных стран.



Поскольку у нас в наличии лог-события за период с мая по июнь 2014 года, установите соответствующий диапазон дат в фильтре времени. Перейдите в меню Time Filter->Absolute Time Range (Фильтр времени->Абсолютный диапазон времени) и укажите в поле From (От) значение 2014-05-28 00:00:00.000 и в поле To (До) – 2014-07-01 00:00:00.000. Нажмите кнопку Go (Начать).

Анализ кодов ответов по времени

Данная визуализация может быть создана в виде гистограммы.

Создайте новую визуализацию.

1. Нажмите кнопку **New** (Создать) и выберите **Vertical Bar** (Гистограмма).
2. Выберите **Logstash-*** в поле **From a New Search, Select Index** (Из нового поиска, выбрать индекс).
3. На оси X выберите **Date Histogram** (Гистограмма даты) и значение **@timestamp**.
4. Нажмите кнопку **Add sub-buckets** (Добавить субсегменты) и выберите **Split Series** (Раздельная серия).
5. Выберите **Terms** (Термы) в поле **Sub Aggregation** (Субагрегация).
6. Выберите **response.keyword** в **Field** (Поле).
7. Нажмите кнопку **Play** (Воспроизвести) (применив изменения).

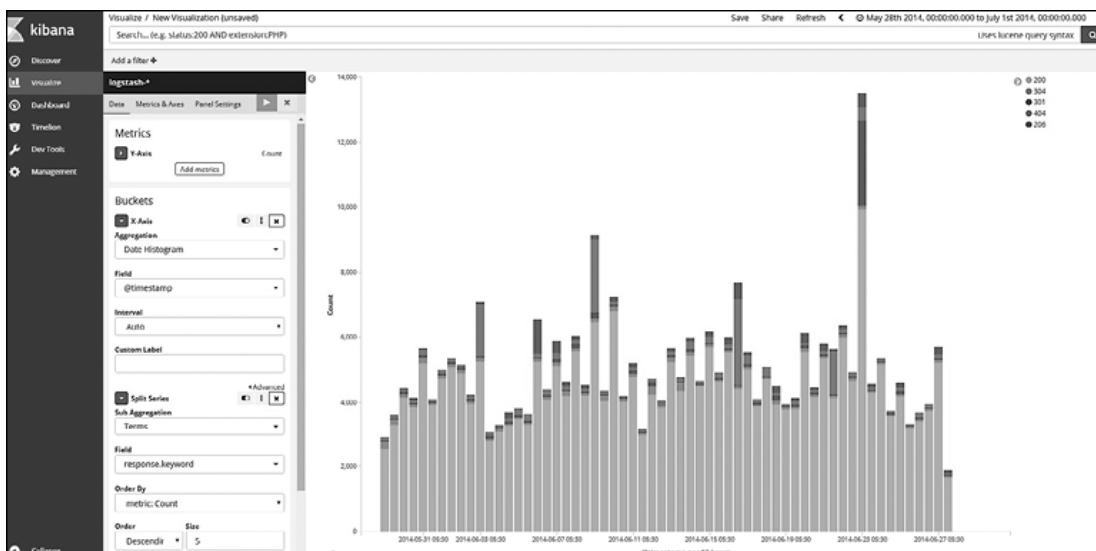


Рис. 7.35

Сохраните визуализацию под названием «Анализ кодов ответов по времени».

Как можно видеть на рис. 7.35, в некоторые дни, например 9, 16 июня и т.д., часто выдается ошибка 404. Теперь для анализа событий 404 щелкните на значении **404** на панели меток/ключей и далее выберите **positive filter** (положительный фильтр) (рис. 7.36).

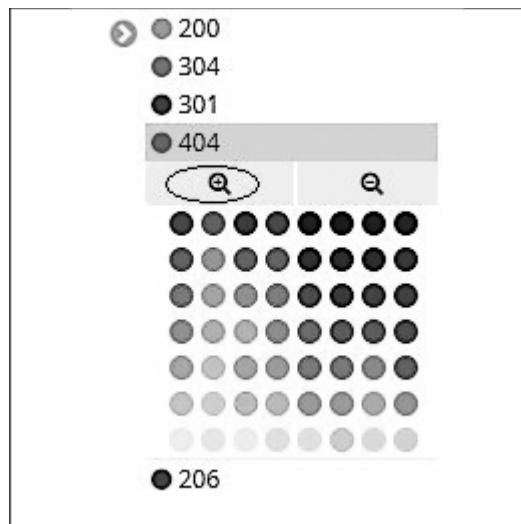


Рис. 7.36

В результате вы получите диаграмму следующего вида (рис. 7.37).

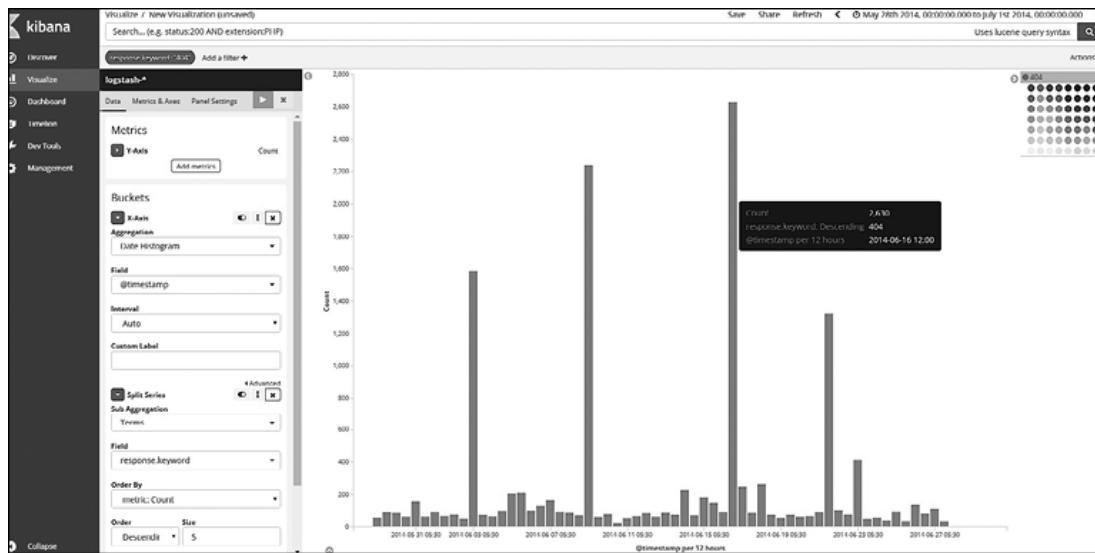


Рис. 7.37



Вы можете развернуть метки/ключи и выбрать цвета на палитре, тем самым изменив цветовое оформление визуализации. Закрепите фильтр и перейдите на страницу Discover (Исследование), чтобы увидеть запросы, которые приводят к ошибке 404.

Поиск топ-10 запрошенных ссылок

Для этого удобно использовать визуализацию в виде таблицы данных (рис. 7.38).

Выполните следующие шаги.

1. Создайте новую визуализацию.

2. Нажмите кнопку **New** (Создать) и выберите **Data Table** (Таблица данных).
3. Выберите **Logstash-*** в поле **From a New Search, Select Index** (Из нового поиска, выбрать индекс).
4. Выберите тип сегментов **Split Rows** (Разделенные строки).
5. Выберите в поле **Aggregation** (Агрегация) значение **Terms** (Термы).
6. Выберите **response.keyword** в **Field** (Поле).
7. Установите размер **10**.
8. Нажмите кнопку **Play** (Воспроизвести) (применив изменения).



Рис. 7.38

Сохраните визуализацию под названием «Поиск топ-10 запрошенных ссылок».



Для получения осмысленных названий в результатах агрегации можно использовать поля Custom Label (Пользовательские метки). Большинство визуализаций поддерживают такие поля. Визуализацию в виде таблицы данных можно экспортировать как файл .csv, щелкнув на ссылках Raw (Необработанный) или Formatted (Форматированный), которые находятся под визуализацией.

Анализ использования сетевого трафика для каждой из топ-5 стран по времени

Выполните следующие шаги.

1. Создайте новую визуализацию (рис. 7.39).
2. Нажмите кнопку **New** (Создать) и выберите **Area Chart** (Диаграмма области).
3. Выберите **Logstash-*** в поле **From a New Search, Select Index** (Из нового поиска, выбрать индекс).
4. На оси **Y (Y axis)** выберите **Sum** (Сумма) в качестве типа агрегации и **bytes** (байты) как поле.
5. На оси **X (X axis)** выберите **Date Histogram** (Гистограмма даты) и **@timestamp** в **Field** (Поле).
6. Нажмите кнопку **Add sub-buckets** (Добавить субсегменты) и выберите **Split Series** (Раздельная серия).

7. Выберите **Terms** (Термы) в поле **Sub Aggregation** (Субагрегация).
8. Выберите **geoip.country_name.keyword** в **Field** (Поле).
9. Нажмите кнопку **Play** (Воспроизвести) (применив изменения).

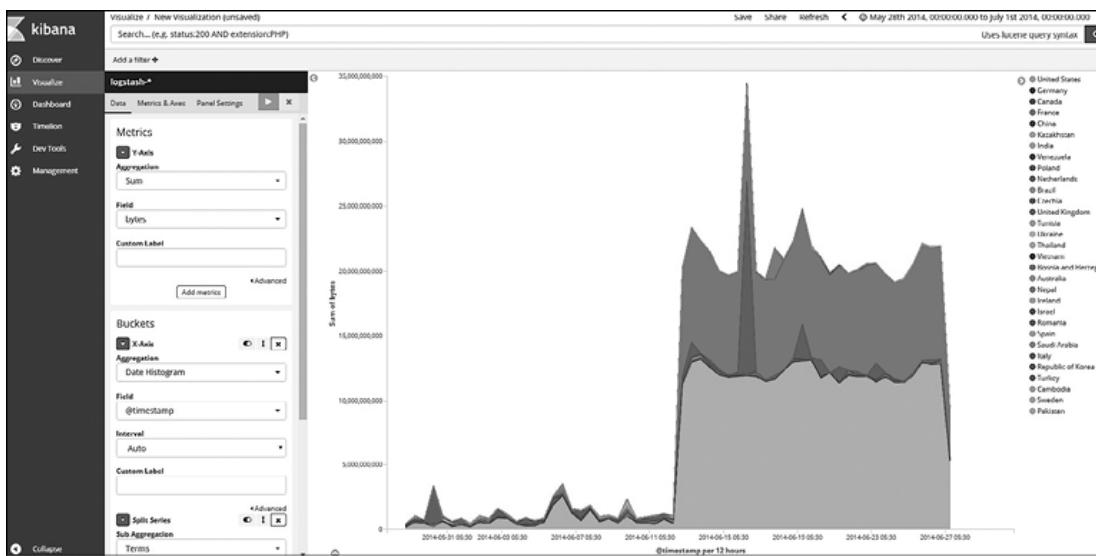


Рис. 7.39

Сохраните визуализацию под названием «Анализ использования сетевого трафика для каждой из топ-5 стран».

Что, если нужно найти не только первую пятерку стран? Перестройте порядок агрегаций и нажмите **Play** (Воспроизвести) (рис. 7.40).

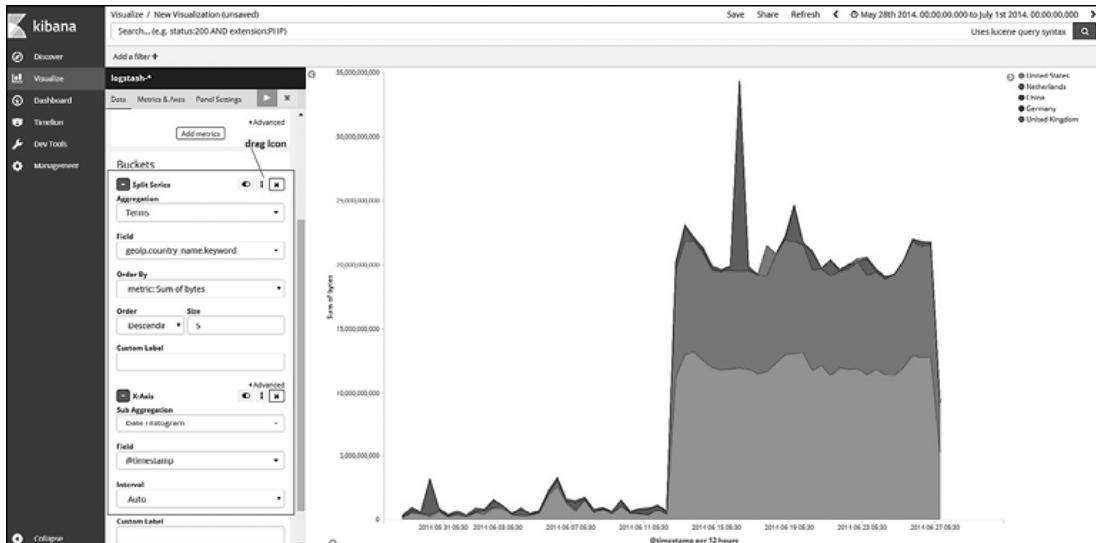


Рис. 7.40



Порядок агрегаций имеет значение.

Анализ веб-трафика из различных стран

Нужные данные можно легко визуализировать с помощью карты координат.

Выполните следующие шаги.

1. Создайте новую визуализацию (рис. 7.41).
2. Нажмите кнопку **New** (Создать) и выберите **Coordinate Map** (Карта координат).
3. Выберите **Logstash-*** в поле **From a New Search, Select Index** (Из нового поиска, выбрать индекс).
4. Выберите тип сегментов **Geo Coordinates** (Геокоординаты).

5. Выберите агрегацию **Geohash**.
6. Выберите поле **geoip.location**.
7. На вкладке параметров выберите в поле **Map Type** (Тип карты) значение **Heatmap** (Карта температур).
8. Нажмите кнопку **Play** (Воспроизвести) (применив изменения).

Сохраните визуализацию как «Анализ веб-трафика из различных стран». Согласно этой визуализации, наибольшая часть трафика происходит из Калифорнии.

Для той же визуализации при измерении в *байтах* результат изменится следующим образом (рис. 7.42).

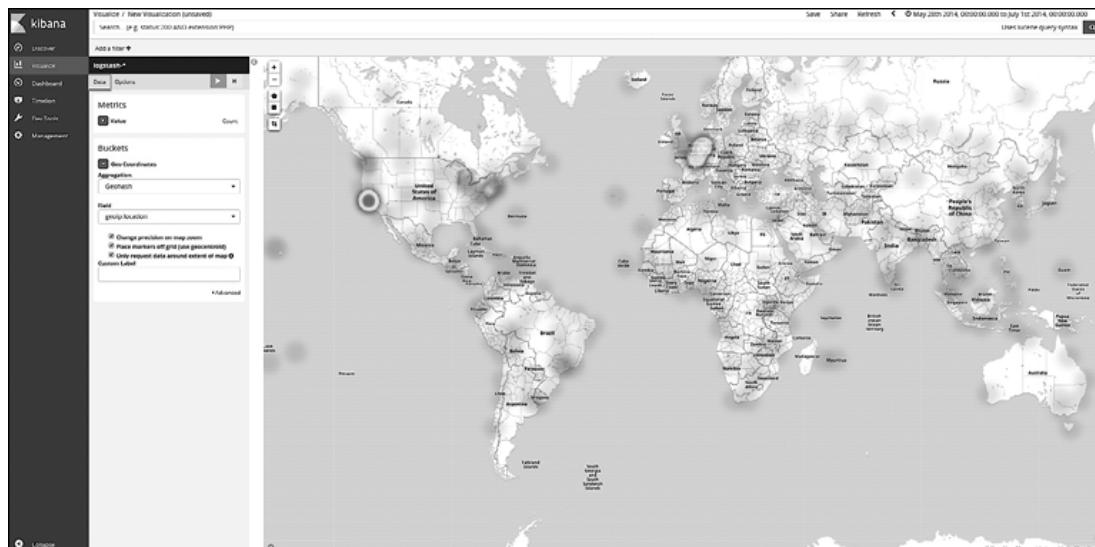


Рис. 7.41

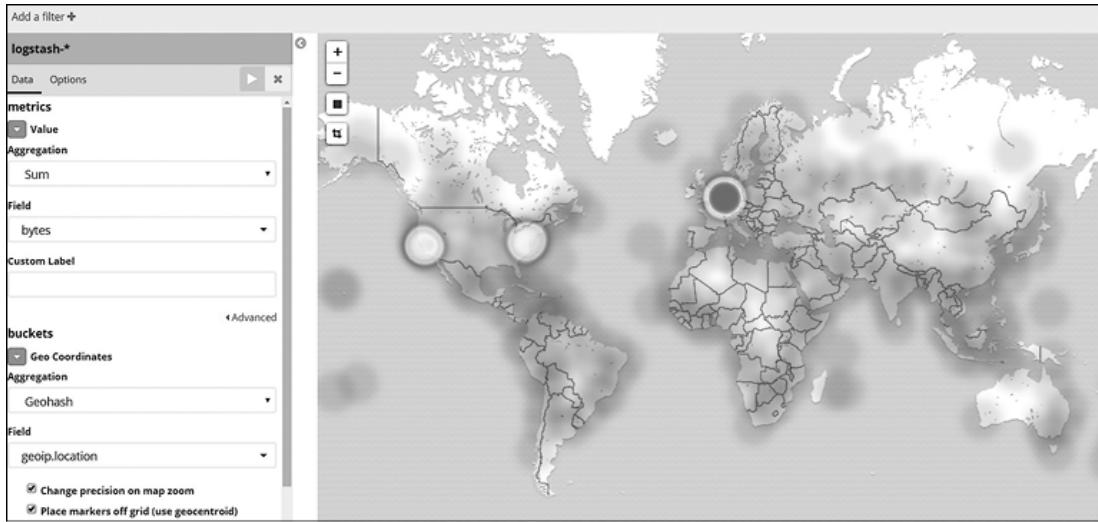


Рис. 7.42



Для изменения масштаба карты вы можете нажимать кнопки $+$ / $-$, расположенные слева вверху от карты.

Для выбора области фильтрации документов можно нажать кнопку Draw Rectangle (Начертить прямоугольник), которая находится слева вверху, под кнопками изменения масштаба карты. После этого вы можете закрепить фильтр и перейти на страницу Discover (Исследование) для рассмотрения документов, принадлежащих выбранной области.

Поиск наиболее популярного пользовательского агента

Нужные данные можно легко визуализировать с помощью разных диаграмм. Выберем облако тегов.

Выполните следующие шаги.

1. Создайте новую визуализацию (рис. 7.43).

2. Нажмите кнопку **New** (Создать) и выберите **Tag Cloud** (Облако тегов).
3. Выберите **Logstash-*** в поле **From a New Search, Select Index** (Из нового поиска, выбрать индекс).
4. Выберите тип сегментов **Tags** (Теги).
5. Выберите **Terms** (Термы) в поле **Aggregation** (Агрегация).
6. Выберите поле **useragent.name.keyword**.
7. Установите размер **10** и нажмите кнопку **Play** (Воспроизвести) (применив изменения).

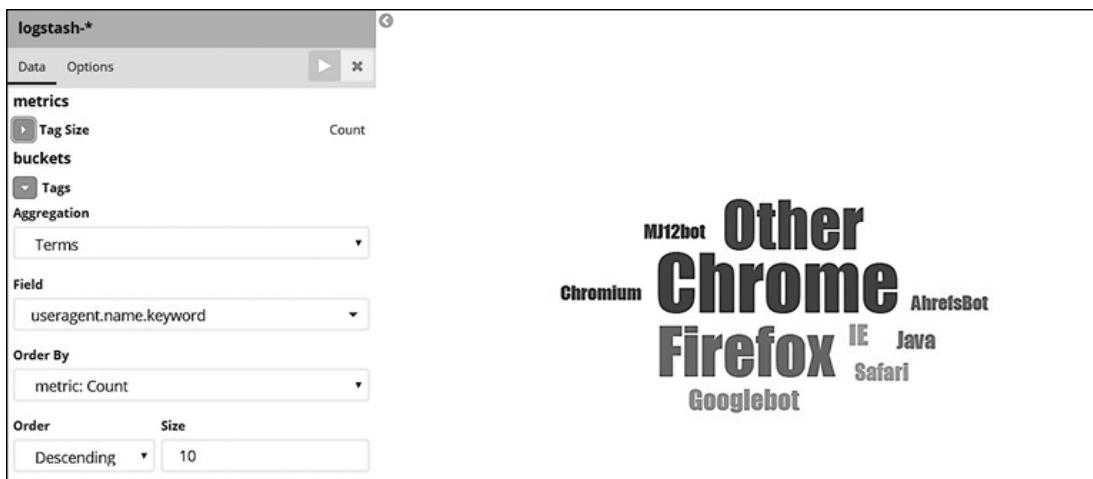


Рис. 7.43

Сохраните визуализацию под названием «Поиск наиболее популярного пользовательского агента». В списке пользовательских агентов для наших данных трафика самым популярным является Chrome, после него — Firefox.

Панели управления

Панели управления помогают собирать разные визуализации на одной странице. Вы можете настроить панель управления, используя ранее сохраненные визуализации и запросы.

Простая панель управления будет выглядеть следующим образом (рис. 7.44).

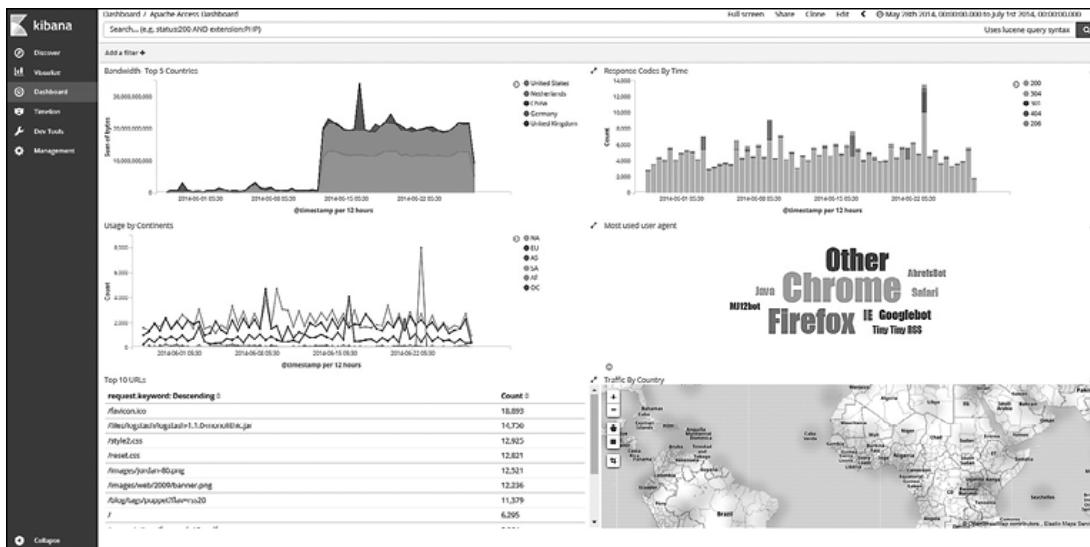


Рис. 7.44

Рассмотрим, как создать панель управления для нашего примера с анализом лог-данных.

Создание панели управления

Для создания новой панели управления перейдите на страницу **Dashboard** (Панель управления) и нажмите кнопку **Create a dashboard** (Создать панель управления) или + (рис. 7.45).

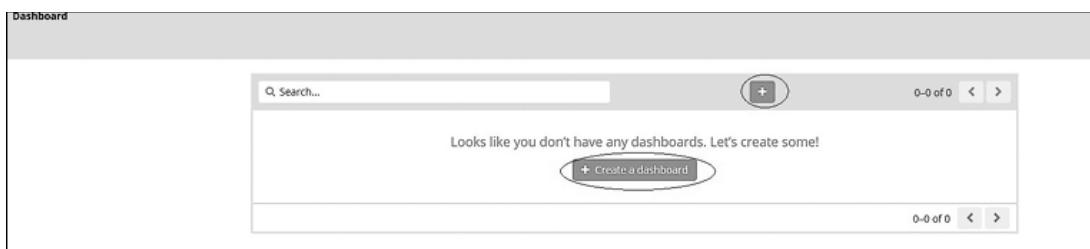


Рис. 7.45

На следующей странице вы можете нажать кнопку **Add** (Добавить) (рис. 7.46), после чего получите список доступных сохраненных визуализаций и результатов поиска, которые доступны для добавления. Щелчок на сохраненных результатах/визуализациях приведет к добавлению их на страницу (рис. 7.47).

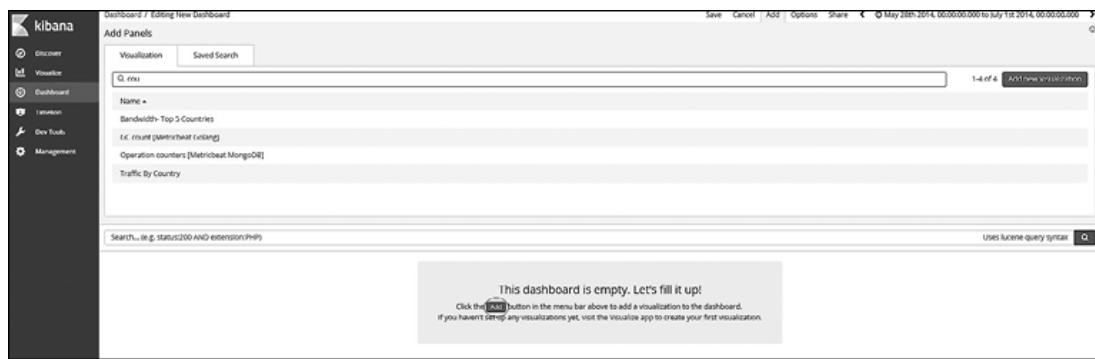


Рис. 7.46



Рис. 7.47



Рис. 7.48

Есть возможность разворачивать, редактировать визуализации, менять их порядок или удалять, используя кнопки, размещенные в верхнем углу каждой визуализации (рис. 7.48).



Используя панель запросов, фильтры полей и времени, вы можете фильтровать результаты поиска. Панель управления будет реагировать на эти изменения, если они касаются встроенных в нее визуализаций.

К примеру, возможно, вам необходимо знать только первые несколько значений пользовательских агентов и устройств по стране при условии, что код ответа – 404.

Более детально панель запросов, фильтры полей и времени рассматривались в разделе «Исследование» выше.

Сохранение панели управления

Как только все необходимые визуализации добавлены на панель управления, убедитесь в том, что вы ее сохранили. Для этого нажмите кнопку **Save** (Сохранить) на панели инструментов и введите название. При сохранении панели управления также сохраняются все критерии запроса и фильтры. Если вы хотите сохранить и фильтры времени, установите флажок **Store time with dashboard** (Сохранить время на панели управления). Это может быть полезным, если вы планируете в будущем вернуться к панели управления в ее текущем состоянии (рис. 7.49).



Рис. 7.49

Клонирование панели управления

Используя функцию клонирования, вы можете скопировать текущую панель управления вместе с ее запросами и фильтрами и создать новую. Например, вы можете создать новые панели управления для стран или континентов (рис. 7.50).

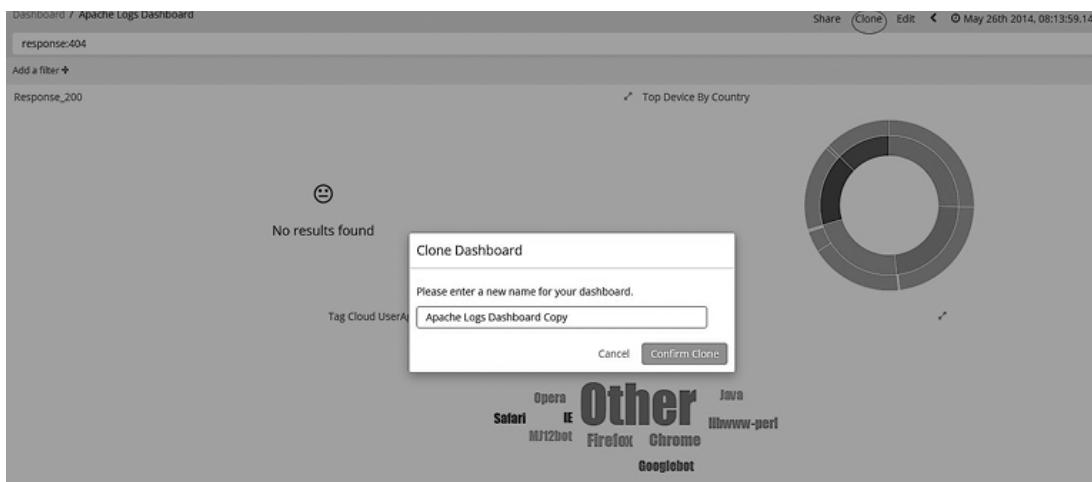


Рис. 7.50



Фоновое оформление панели управления может быть изменено со светлого на темное. При нажатии кнопки **Edit** (Редактировать) на панели инструментов вы увидите кнопку **Options** (Опции), которая и позволяет добавить эту возможность.

Общий доступ к панели управления

Используя функцию **Share** (Поделиться), вы можете отправить прямую ссылку на панель управления Kibana другому пользователю или встроить панель в веб-страницу в виде Iframe (рис. 7.51).



Рис. 7.51

Timelion

Timelion — инструмент визуализации для анализа временного ряда в Kibana. Он позволяет комбинировать абсолютно независимые визуализации в пределах одной. Используя простой язык выражений, вы можете выполнять сложные математические вычисления, такие как деление и вычитание метрик, расчет производных и скользящего среднего, а также визуализировать эти вычисления.

Пользовательский интерфейс Timelion

Timelion находится на левой панели пользовательского интерфейса Kibana, между значками разделов **Dashboard** (Панель управления) и **Dev Tools** (Инструменты разработчика) (рис. 7.52).



Рис. 7.52

Основной компонент пользовательского интерфейса Timelion — **Timelion query bar** (Панель запросов Timelion), которая позволяет определять выражения, влияющие на генерирование диаграмм. Вы можете задать несколько выражений, разделяя их запятыми, а также объединять функции в цепочку.

Пользовательский интерфейс Timelion также предлагает следующие функции.

- **New (Создать).** Используется с целью создания нового листа Timelion для формирования диаграмм.
- **Add (Добавить).** С помощью этой функции вы можете создать несколько диаграмм на одном листе Timelion.
- **Save (Сохранить).** Используется для сохранения страницы Timelion. Есть два варианта: сохранить как лист Timelion или сохранить текущее выражение как панель управления Kibana.

- **Open (Открыть).** Применяется для открытия существующего листа Timelion.
- **Options (Опции).** Здесь вы можете указать количество строк и столбцов в листе Timelion.
- **Docs (Документы).** Документация для начала работы с Timelion, а также документация обо всех поддерживаемых функциях выражений Timelion.
- **Time Filter (Фильтр времени).** Параметры фильтра времени для фильтрации данных.

Выражения Timelion

Простейшее выражение Timelion для создания диаграмм выглядит следующим образом:

```
.es(*)
```

Выражения Timelion всегда начинаются с точки, далее идет название функции, которая принимает один или несколько параметров. Выражение `.es(*)` запрашивает данные из всех индексов в Elasticsearch. По умолчанию количество документов будет подсчитано и показано на диаграмме с обзором по времени.

Если вы хотите ограничить работу Timelion с данными из определенного индекса (например, `logstash-*`), то можете указать индекс внутри функции следующим образом:

```
.es(index=logstash-*)
```

Поскольку Timelion является визуализатором временного ряда, он использует поле `@timestamp`, находящееся в индексе,

как время для размещения данных на оси X. Вы можете изменить это, подставив необходимое поле времени как значение параметра `timefield`.

В Timelion присутствует полезная функция автозавершения, которая поможет вам быстрее формировать выражения (рис. 7.53).



Рис. 7.53

Рассмотрим еще несколько примеров использования Timelion на практике.



Поскольку у нас в наличии лог-события за период с мая по июнь 2014 года, установите соответствующий диапазон дат в фильтре времени. Перейдите в меню Time Filter—>Absolute Time Range (Фильтр времени—>Абсолютный диапазон времени) и укажите в поле From (От) значение 2014-05-28 00:00:00.000 и в поле To (До) – 2014-07-01 00:00:00.000. Нажмите кнопку Go (Начать).

Найдем средний расход байтов в США по времени (рис. 7.54). Выражение будет выглядеть следующим образом:

```
.es(q='geoip.country_code3:US',metric='avg:bytes')
```

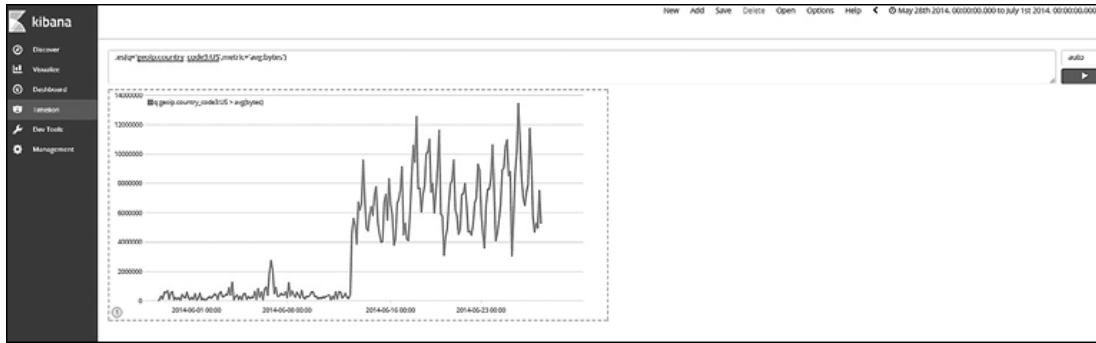


Рис. 7.54

Timelion позволяет размещать несколько диаграмм на одной странице. Для этого нужно разделить выражения запятыми.

Найдем средний расход байтов в США и Китае по времени (рис. 7.55). Выражение будет выглядеть следующим образом:

```
.es(q='geoip.country_code3:US',metric='avg:bytes'),
.es(q='geoip.country_code3:CN',metric='avg:bytes')
```

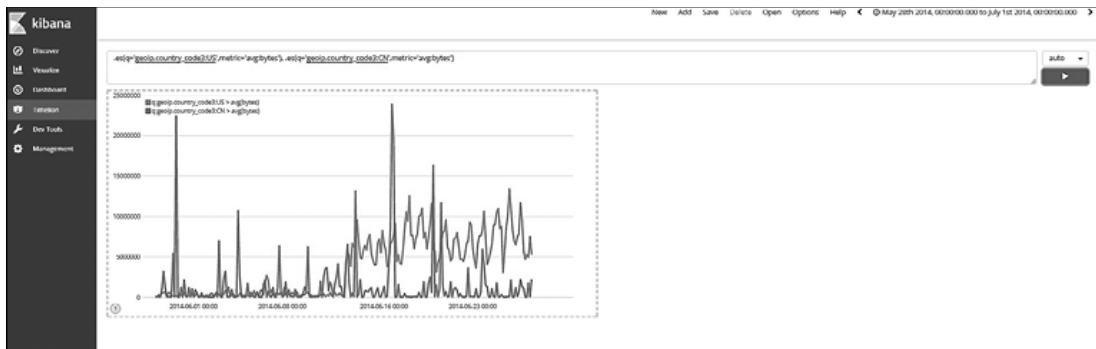


Рис. 7.55

Timelion также позволяет изменять функции. Изменим метку и цвет предыдущих диаграмм (рис. 7.56). Выражение будет выглядеть следующим образом:

```
.es(q='geoip.country_code3:US',metric='avg:bytes').label('United States').color('yellow'),
```

```
.es(q='geoip.country_code3:CN',metric='avg:bytes').label('China').color('red')
```

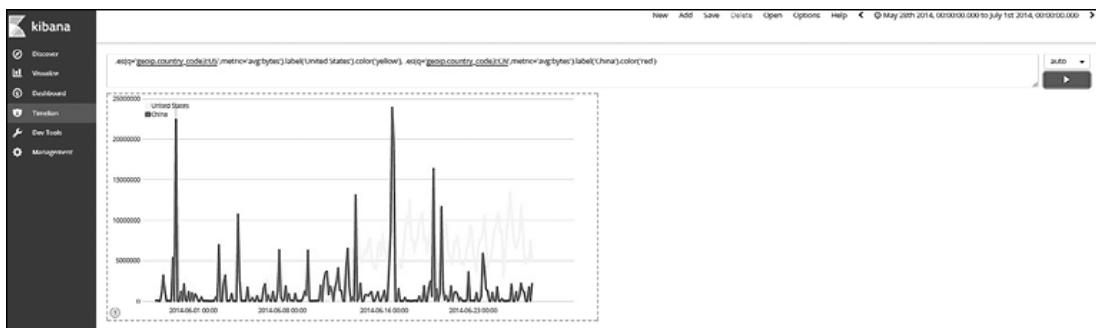


Рис. 7.56

Еще одной полезной функцией является использование смещений при анализе старых данных. Это полезно при сравнении текущих значений с более ранними для анализа динамики. Сравним сумму использования байтов с показателями за предыдущую неделю для США (рис. 7.57). Выражение будет выглядеть следующим образом:

```
.es(q='geoip.country_code3:US',metric='sum:bytes').label('Current Week'),  
.es(q='geoip.country_code3:US',metric='sum:bytes',offset=-1w).label('Previous Week')
```



Предыдущий скриншот показывает возможность добавления нескольких диаграмм на один лист Timelion. Для этого необходимо нажать кнопку Add (Добавить). Выбор диаграмм изменяет ассоциированное выражение на панели запросов Timelion (Timelion Query Bar).

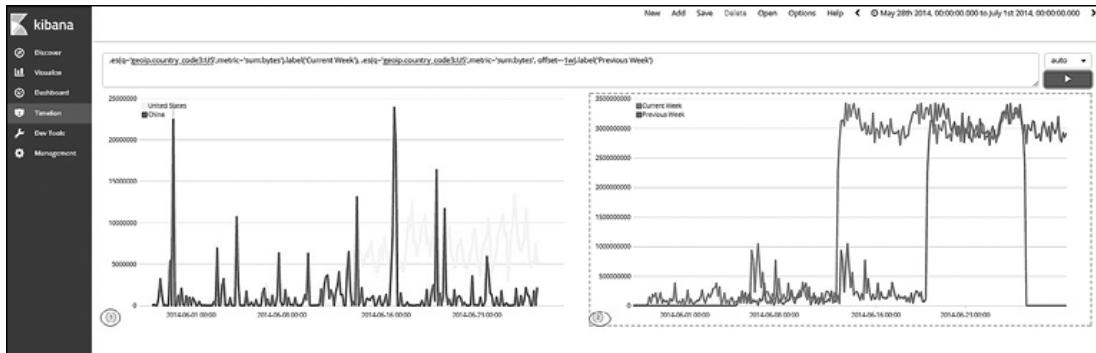


Рис. 7.57

В Timelion также поддерживается использование данных из внешних источников с помощью публичного API. В комплекте вы найдете родной API для получения данных Всемирного банка, Quandl и Graphite.



Выражения Timelion поддерживают более 50 различных функций, которые вы можете использовать для создания выражений (более детально читайте по ссылке <https://github.com/elastic/timelion/blob/master/FUNCTIONS.md>).

Использование плагинов

Плагины предоставляют отличную возможность расширить функционал Kibana. Все установленные плагины сохраняются в папке `$KIBANA_HOME/plugins`. Компания — производитель Kibana — Elastic — предоставляет множество плагинов, также доступны сторонние плагины, которыми вы можете воспользоваться.

Установка плагинов

Перейдите в папку KIBANA_HOME и выполните команду `install`, как показано в следующем фрагменте кода. Таким образом устанавливаются любые плагины. Во время установки необходимо указать либо название плагина (если он от компании Elastic), либо URL-ссылку, по которой размещен плагин:

```
$ KIBANA_HOME>bin/kibana-plugin install <имя  
пакета или URL>
```

Например, для установки x-pack, плагина от компании Elastic, выполните следующую команду:

```
$ KIBANA_HOME>bin/kibana-plugin install x-pack
```

Для установки общедоступного плагина, например LogTrail (<https://github.com/sivasamyk/logtrail>), выполните такую команду:

```
$ KIBANA_HOME>bin/kibana-plugin install  
https://github.com/sivasamyk/logtrail/releases/  
download/v0.1.23/logtrail-6.0.0-0.1.23.zip
```



LogTrail – это плагин для просмотра, анализа, поиска и отслеживания лог-событий из различных источников в реальном времени с удобным для разработчиков интерфейсом. Создан компанией Papertrail (<https://papertrailapp.com/>).



Список общедоступных плагинов Kibana вы найдете по ссылке <https://www.elastic.co/guide/en/kibana/6.0/known-plugins.html>.

Удаление плагинов

Для удаления плагина перейдите в папку `KIBANA_HOME` и выполните команду `remove`, указав название плагина:

```
$ KIBANA_HOME>bin/kibana-plugin remove x-pack
```

Резюме

В этой главе вы узнали, как эффективно использовать Kibana для создания красивых панелей управления, которые помогают в исследовании данных.

Вы научились настраивать Kibana для визуализации данных из Elasticsearch. Кроме того, разобрались, как добавлять плагины.

В следующей главе мы поговорим об Elasticsearch и ее ключевых компонентах, позволяющих создавать контейнеры данных. Вы также узнаете о визуализации данных и расширениях, которые необходимы для отдельных сценариев использования.

8. Elastic X-Pack

X-Pack — расширение для Elastic Stack, которое предоставляет настройки безопасности, уведомлений, мониторинга, отчетности, машинного обучения, а также возможности построения графиков в одном, легко устанавливаемом пакете. X-Pack добавляет важные функции, необходимые для подготовки Elastic Stack к эксплуатации. В отличие от других компонентов Elastic Stack с открытым исходным кодом, X-Pack является коммерческим предложением от Elastic.co, для его работы требуется платная лицензия. При первой установке вы получаете версию на пробный период 30 дней. Базовая, или бесплатная, версия предоставляет только возможность мониторинга и инструменты разработчика, такие как профайлер поиска (Search Profiler) или отладчик Grok (Grok Debugger). Несмотря на то что X-Pack поставляется как пакет, вы можете легко добавлять/исключать те функции, которые хотите/не хотите использовать.

В этой главе мы рассмотрим следующие темы.

- Установка X-Pack в Elasticsearch или Kibana.
- Настройка безопасности в Elasticsearch или Kibana.
- Мониторинг Elasticsearch.
- Работа с уведомлениями.

Установка X-Pack

Поскольку X-Pack является расширением Elasticsearch, до его установки необходимо инсталлировать Elasticsearch и Kibana.

Следует запускать версию X-Pack, которая совпадает с версией Elasticsearch и Kibana.

Установка X-Pack в Elasticsearch

X-Pack устанавливается так же, как и любой другой плагин Elasticsearch.

Ниже перечислены шаги для установки X-Pack в Elasticsearch.

1. Перейдите в папку ES_HOME.
 2. Установите X-Pack, используя следующую команду:

```
$ ES_HOME> bin/elasticsearch-plugin install x-pack
```

Во время установки вам будет задан вопрос о предоставлении X-Pack дополнительных разрешений, которые необходимы для отправки уведомлений по почте, а также для запуска аналитического движка машинного обучения Elasticsearch. Для продолжения введите Y, для отмены установки — N.

В процессе установки вы получите следующие логи/подсказки:

```
-> Downloading x-pack from elastic
[=====]
=====] 100%
@     WARNING: plugin requires additional
permissions @
```

aa
aaaaaaaaaaaaaaaaaaaaaa

* java.io.FilePermission \\\.\pipe*
read,write

* java.lang.RuntimePermission
accessClassInPackage.com.sun.activation.reg
istries

* java.lang.RuntimePermission
getClassLoader

* java.lang.RuntimePermission
setContextClassLoader

* java.lang.RuntimePermission setFactory

* java.net.SocketPermission *
connect,accept,resolve

* java.security.SecurityPermission
createPolicy.JavaPolicy

* java.security.SecurityPermission
getPolicy

* java.security.SecurityPermission
putProviderProperty.BC

* java.security.SecurityPermission
setPolicy

* java.util.PropertyPermission * read,write

* java.util.PropertyPermission
sun.nio.ch.bugLevel write

See

[http://docs.oracle.com/javase/8/docs/techno
tes/guides/security/
permissions.html](http://docs.oracle.com/javase/8/docs/techno
tes/guides/security/
permissions.html) for descriptions of what
these permissions allow and the associated
risks.

Continue with installation? [y/N]y

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
@ WARNING: plugin forks a native controller  
@  
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa  
aaaaaaaaaaaaaaaaaaaa  
This plugin launches a native controller  
that is not subject to the Java security  
manager nor to system call filters.
```

Continue with installation? [y/N]y

Elasticsearch keystore is required by
plugin [x-pack], creating...

-> **Installed x-pack**

3. Перезапустите Elasticsearch:

```
$ ES_HOME> bin/elasticsearch
```

4. Установите пароли для пользователя по умолчанию и
резервного: elastic, kibana и logstash_system,
выполнив команду:

```
$          ES_HOME>bin/x-pack/setup-passwords  
interactive
```

Вы должны получить следующие логи для ввода пароля
резервного/основного пользователя:

```
Initiating the setup of reserved user  
elastic,kibana,logstash_system passwords.  
You will be prompted to enter passwords as  
the process progresses.
```

```
Please confirm that you would like to  
continue [y/N]y  
Enter password for [elastic]: elastic  
Reenter password for [elastic]: elastic  
Enter password for [kibana]: kibana  
Reenter password for [kibana]: kibana  
Enter password for [logstash_system]:  
logstash  
Reenter password for [logstash_system]:  
logstash  
Changed password for user [kibana]  
Changed password for user [logstash_system]  
Changed password for user [elastic]
```



Обратите внимание на установленные пароли. Вы можете выбрать любой пароль по своему желанию. Мы установили следующие пароли: elastic, kibana и logstash – для пользователей elastic, kibana и logstash_system и будем применять их на протяжении этой главы.

Для проверки установки X-Pack и безопасности откройте в своем браузере Elasticsearch по адресу <http://localhost:9200/>. Вы должны увидеть поле авторизации Elasticsearch. Можете указать здесь встроенного пользователя **elastic** и пароль **elastic**. После успешной регистрации вы увидите следующий ответ:

```
{  
name: "fwDdHSI",  
cluster_name: "elasticsearch",  
cluster_uuid: "08wSPsjSQCmeRaxF4iHizw",
```

```
version: {  
  number: "6.0.0",  
  build_hash: "8f0685b",  
  build_date: "2017-11-10T18:41:22.859Z",  
  build_snapshot: false,  
  lucene_version: "7.0.1",  
  minimum_wire_compatibility_version: "5.6.0",  
  minimum_index_compatibility_version: "5.0.0"  
},  
tagline: "You Know, for Search"  
}
```

Типичный кластер Elasticsearch состоит из нескольких узлов, следовательно, X-Pack необходимо установить на каждом узле кластера.



Для пропуска подсказок установки используйте параметр -batch: \$ES_HOME>bin/elasticsearch-plugin install x-pack --batch.



При установке X-Pack были созданы следующие папки: x-pack в bin, config и plugins в ES_HOME. Мы рассмотрим их в следующих разделах этой главы.

Установка X-Pack в Kibana

X-Pack устанавливается аналогично любому другому плагину расширения Kibana.

Далее перечислены шаги для установки X-Pack в Kibana.

1. Перейдите в папку `KIBANA_HOME`.

2. Установите X-Pack, используя следующую команду:

```
$KIBANA_HOME>bin/kibana-plugin install x-pack
```

В процессе установки вы получите следующие логи/подсказки:

```
Attempting to transfer from x-pack
Attempting to transfer from
https://artifacts.elastic.co/downloads/kibana-plugins/x-pack/x-pack
-6.0.0.zip
Transferring 120307264
bytes.....
Transfer complete
Retrieving metadata from plugin archive
Extracting plugin archive
Extraction complete
Optimizing and caching browser bundles...
Plugin installation complete
```

3. Добавьте следующую информацию авторизации в файл `kibana.yml`, который находится в папке `$KIBANA_HOME/config`, и сохраните его:

```
elasticsearch.username: "kibana"
```

`elasticsearch.password: "kibana"`



Если во время настройки паролей вы выбрали другой пароль для пользователя kibana, указывайте это значение для `elasticsearch.password`.

4. Запустите Kibana.

\$KIBANA_HOME>bin/kibana

Для проверки установки X-Pack перейдите по ссылке <http://localhost:5601/> для открытия Kibana. Вы должны перейти на страницу регистрации (рис. 8.1). Для авторизации можете указать встроенного пользователя **elastic** и пароль **elastic**.

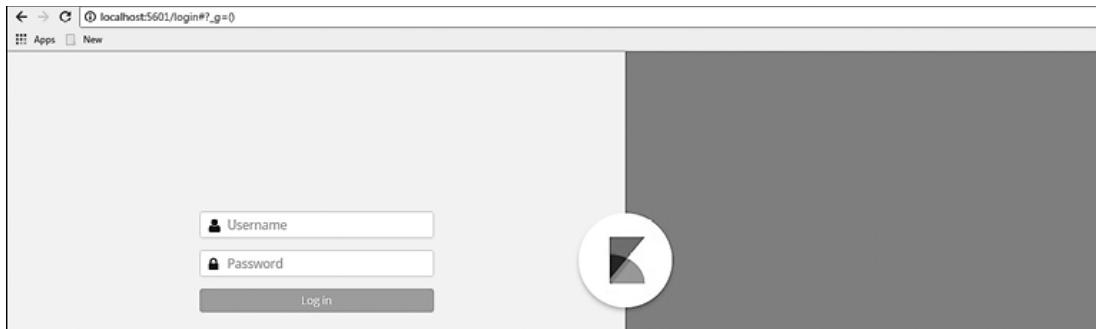


Рис. 8.1



При установке X-Pack была создана папка с названием x-pack, расположенная в каталоге с плагинами KIBANA_HOME.

Можете установить X-Pack и в Logstash. Однако в данный момент X-Pack поддерживает только мониторинг Logstash.

Удаление X-Pack

Для удаления X-Pack выполните следующие шаги.

1. Остановите работу Elasticsearch.

2. Удалите X-Pack из Elasticsearch:

```
$ES_HOME>bin/elasticsearch-plugin remove x-pack
```

3. Перезапустите Elasticsearch и остановите работу Kibana.

Удалите X-Pack из Kibana:

```
$KIBANA_HOME>bin/kibana-plugin remove x-pack
```

4. Перезапустите Kibana.

Настройка X-Pack

Сразу после установки можно использовать встроенные возможности X-Pack, относящиеся к безопасности, уведомлениям, мониторингу, отчетности, машинному обучению, а также к построению графиков. Возможно, вам не потребуется весь возможный функционал этого расширения, поэтому вы можете выбирать необходимые инструменты, включать или выключать их в конфигурационных файлах `elasticsearch.yml` и `kibana.yml`.

Elasticsearch поддерживает следующие функции и настройки в файле `elasticsearch.yml`.

Функция	Настройка	Описание
Машинное обучение	<code>xpack.ml.enabled</code>	Установите отрицательное значение для отключения функционала машинного обучения X-Pack
Мониторинг	<code>xpack.monitoring.enabled</code>	Установите отрицательное значение для отключения функционала мониторинга Elasticsearch
Безопасность	<code>xpack.security.enabled</code>	Установите отрицательное значение для отключения функционала безопасности X-Pack
Наблюдатель	<code>xpack.watcher.enabled</code>	Установите отрицательное значение для отключения функционала наблюдателя

Kibana поддерживает следующие функции и настройки в файле `kibana.yml`.

Функция	Настройка	Описание
Машинное обучение	<code>xpack.ml.enabled</code>	Установите отрицательное значение для отключения функционала машинного обучения X-Pack
Мониторинг	<code>xpack.monitoring.enabled</code>	Установите отрицательное значение для отключения функционала мониторинга Kibana
Безопасность	<code>xpack.security.enabled</code>	Установите отрицательное значение для отключения функционала безопасности X-Pack
Построение диаграмм	<code>xpack.graph.enabled</code>	Установите отрицательное значение для отключения функционала диаграмм X-Pack
Отчетность	<code>xpack.reporting.enabled</code>	Установите отрицательное значение для отключения функционала отчетности X-Pack



Если X-Pack установлен в Logstash, можете отключить мониторинг, присвоив параметру `xpack.monitoring.enabled` значение `false` в конфигурационном файле `logstash.yml`.

Безопасность

Компоненты Elastic Stack не имеют встроенных механизмов защиты — кто угодно может получить к ним доступ. Это представляет угрозу безопасности при внедрении Elastic Stack в эксплуатацию. Для предотвращения несанкционированного доступа применяются разные механизмы обеспечения безопасности, такие как запуск Elastic Stack вместе с системой сетевой защиты (файерволлом) и использование обратных прокси (nginx, HAProxy и пр.). Компания Elastic.co предлагает коммерческий продукт для защиты Elastic Stack. Он поставляется как часть пакета X-Pack и называется Security.

Модуль безопасности X-Pack обеспечивает следующие способы защиты Elastic Stack:

- пользовательскую аутентификацию и авторизация;
- аутентификацию узла/клиента и шифрование канала;
- аудит.

Пользовательская аутентификация

Пользовательская аутентификация — процесс предотвращения несанкционированного доступа к кластеру Elastic Stack путем проверки данных пользователя. В модуле безопасности X-Pack за процесс аутентификации отвечают один или несколько сервисов, которые называются *областями* (realms). Модуль безопасности предоставляет два типа областей: внутренние и внешние.

Два типа встроенных областей безопасности называются *native* и *file*. Область *native* является областью по умолчанию, данные аутентификации пользователя хранятся в особом индексе *.security-6* в Elasticsearch. Управление

пользователями осуществляется с помощью соответствующего API (User Management API) или на странице **Management** (Управление) в пользовательском интерфейсе Kibana. Мы рассмотрим это детальнее в следующих разделах главы.

Если тип области `file`, тогда данные аутентификации пользователей хранятся в файле в каждом узле. Управление пользователями осуществляется с помощью специальных инструментов, которые предоставляются во время установки X-Pack. Вы можете найти их в папке `$ES_HOME\bin\x-pack`. Там же хранятся и пользовательские файлы. Поскольку сведения о пользователях хранятся в файле, в ответственность администратора входит создание пользователей с одинаковыми данными в каждом узле.

Встроенные внешние области безопасности — это `ldap`, `active_directory` и `pki`, которые используют внешний сервер LDAP, внешний сервер Active Directory и инфраструктуру открытого ключа.

В зависимости от настроенных областей данные аутентификации пользователей необходимо добавлять к запросам в Elasticsearch. Порядок областей, приведенный в файле `elasticsearch.yml`, определяет порядок, в котором области учитываются в процессе аутентификации. Каждая область опрашивается одна за другой в установленном порядке до той поры, пока аутентификация не будет признана успешной. Процесс аутентификации считается успешным при достижении удачной аутентификации запроса одной из областей. Если ни одна из областей не смогла аутентифицировать пользователя, попытка считается неуспешной и клиент получает ошибку аутентификации (HTTP 401).

Если в файле `elasticsearch.yml` не указано никаких областей, тогда по умолчанию используется область типа

`native`. Для задействования областей типа `file` или внешних областей их необходимо указать в файле `elasticsearch.yml`.

Например, в следующем фрагменте кода показана конфигурация цепочки областей, содержащей `native`, `file` и `ldap`:

```
xpack.security.authc:  
  realms:  
    native:  
      type: native  
      order: 0  
    file:  
      type: file  
      order: 1  
    ldap_server:  
      type: ldap  
      order: 2  
      url: 'url_to_ldap_server'
```



Для отключения определенного типа областей безопасности используйте параметр `enabled:false`, как показано в примере ниже:

```
ldap_server:  
  type: ldap  
  order: 2  
  enabled: false  
  url: 'url_to_ldap_server'
```

Авторизация пользователя

После того как пользователь успешно аутентифицирован, включается процесс авторизации. Авторизация определяет, имеет ли пользователь, задающий запрос, достаточно разрешений для выполнения конкретного запроса.

В модуле безопасности X-Pack пользовательская безопасность обеспечивается защищенными ресурсами (*secured resources*). Защищенный ресурс — это ресурс, которому нужен доступ к индексам, документам, полям или доступ на выполнение операций кластеров Elasticsearch. Модуль безопасности X-Pack производит авторизацию, назначая разрешения ролям, которые приписаны пользователям. *Разрешение* — это одна или несколько привилегий относительно защищенного ресурса. *Привилегией* называется группа из одного или нескольких действий, которые пользователь может выполнить по отношению к защищенному ресурсу. Пользователь может иметь одну или несколько ролей, а общий набор разрешений пользователя определяется как сумма всех разрешений во всех ролях (рис. 8.2).

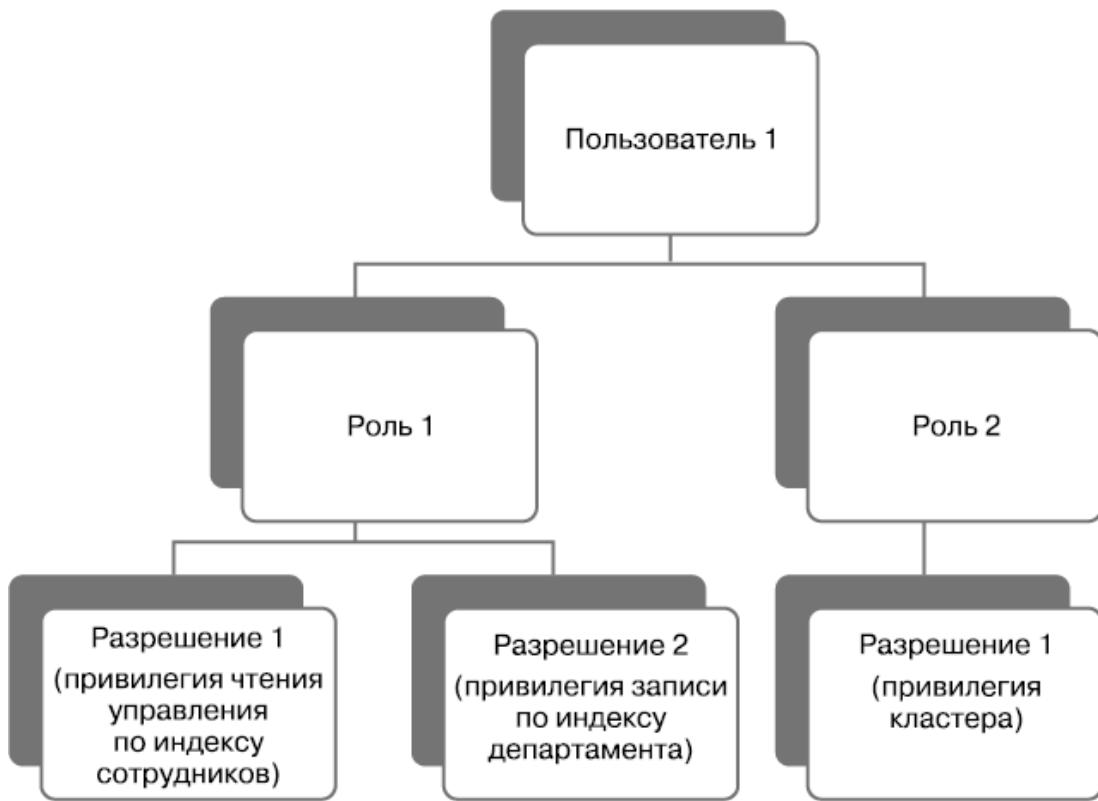


Рис. 8.2

В модуле безопасности X-Pack предоставляются следующие типы привилегий.

- **Привилегии кластера.** Предоставляет доступ к выполнению различных операций в кластере. Возможны такие настройки:
 - `all` — разрешает выполнять все операции администратора кластера: настройку, обновление, переадресацию или управление пользователями и ролями;
 - `monitor` — позволяет выполнять операции только чтения кластера: запрос работоспособности кластера,

состояния кластера, узлов и пр., для проведения мониторинга;

- `manage` — разрешает выполнять операции для обновления кластера, такие как переадресация и обновление настроек кластера.
- **Привилегии индекса.** Предоставляет доступ к выполнению различных операций в индексе:
 - `all` — разрешает выполнять любую операцию в индексе;
 - `read` — позволяет выполнять операции только чтения в индексе;
 - `create_index` — разрешает создать новый индекс;
 - `create` — разрешает индексировать новые документы в индексе;
- **Привилегия выполнения от имени.** Предоставляет возможность действовать от имени пользователя; то есть позволяет аутентифицированному пользователю тестировать права доступа другого пользователя, не зная данных авторизации.



С полным списком привилегий вы можете ознакомиться по ссылке <https://www.elastic.co/guide/en/x-pack/master/security-privileges.html>.

- **Аутентификация узла/клиента и шифрование канала.** Модуль безопасности X-Pack предотвращает сетевые атаки путем шифрования связи. Вы можете шифровать трафик между кластером Elasticsearch и внешними приложениями, а также связь между узлами кластера. Для предотвращения незапланированного подключения узлов к кластеру можно настроить подключение узлов к кластеру только с помощью сертификатов SSL. Доступна также IP-фильтрация для предотвращения незапланированного подключения клиентов приложений, узлов или доставки к кластеру.
- **Аудит.** Аудит позволяет обнаруживать подозрительные действия в кластере. Вы можете использовать его для отслеживания событий, связанных с безопасностью, таких как ошибки аутентификации и отклоненные соединения. Логирование таких событий позволит вам мониторить кластер на предмет подозрительных действий и получать доказательства в случае атаки.

Безопасность на деле

В этом разделе мы рассмотрим создание новых пользователей, новых ролей и ассоциирование ролей с пользователями. Импортируем данные из нашего примера и используем их для того, чтобы разобраться в работе модуля безопасности.

Сохраните следующие данные в файл с названием `data.json`:

```
{"index":  
  {"_index": "employee", "_type": "employee"}}  
{  
  "name": "user1",  
  "email": "user1@packt.com", "salary": 5000,  
  "gender": "M",
```

```

"address1":"312           Main           St",
"address2":"Walthill", "state":"NE"}}
{"index"                           :
{"_index":"employee","_type":"employee"}}
{
"           "name":"user2",
"email": "user2@packt.com", "salary":10000,
"gender": "F",
"address1":"5658      N      Denver      Ave",
"address2":"Portland", "state":"OR"}}
{"index"                           :
{"_index":"employee","_type":"employee"}}
{
"           "name":"user3",
"email": "user3@packt.com", "salary":7000,
"gender": "F",
"address1":"300      Quinterra      Ln",
"address2":"Danville", "state":"CA"}}
{"index"                           :
{"_index":"department","_type":"department"}}
{
"   "name": "IT", "employees":50 }
{"index"                           :
{"_index":"department","_type":"department"}}
{
"   "name": "SALES", "employees":500 }
{"index"                           :
{"_index":"department","_type":"department"}}
{
"   "name": "SUPPORT", "employees":100 }

```



API _bulk позволяет сохранять последнюю строку файла с символом новой строки, \n. При сохранении файла убедитесь, что символ новой строки расположен в последней строке файла.

Перейдите в папку с этим файлом и выполните следующую команду для импортирования данных в Elasticsearch:

```
$ directory_of_data_file> curl -s -H "Content-Type: application/json" -u elastic:elastic -XPOST http://localhost:9200/_bulk --data-binary @data.json
```

Для проверки успешности импорта выполните следующую команду и уточните, правильно ли указано количество документов:

```
D:\packt\book>curl -s -H "Content-Type: application/json" -u elastic:elastic -XGET http://localhost:9200/employee,department/_count
{"count":6,"_shards": {"total":10,"successful":10,"skipped":0,"failed":0}}
```

Создание нового пользователя

В первую очередь зайдите в Kibana как пользователь **elastic** (<http://localhost:5601>).

1. Для создания нового пользователя перейдите в раздел пользовательского интерфейса **Management** (Управление) и выберите **Users** (Пользователи) в разделе **Security** (Безопасность) (рис. 8.3).

The screenshot shows the Kibana Management interface. On the left is a sidebar with icons for Discover, Visualize, Dashboard, Timeline, Machine Learning, Graph, Dev Tools, Monitoring, and Management. The Management icon is highlighted. The main area has a header 'Management' and a sub-header 'Version: 6.0.0'. Below this is a 'Security' section with tabs for 'Users' and 'Roles'. Under 'Users', there is a back arrow icon. To the right of the tabs, there is a 'Watcher' section.

Рис. 8.3

2. На странице **Users** (Пользователи) перечислены все доступные пользователи и их роли. По умолчанию показаны стандартные/резервные пользователи, которые являются частью области безопасности native X-Pack (рис. 8.4).

The screenshot shows the 'Management / Security' section of Kibana. The sidebar includes the 'Management' icon. The main area is titled 'Management / Security' and 'Users Roles'. It features a search bar and a 'Create user' button. A table lists three users: 'elastic' (superuser), 'kibana' (kibana_system), and 'logstash_system' (logstash_system). Each row has a checkbox for 'Full Name' and a 'Reserved' checkbox.

Full Name	Username	Roles	Reserved
<input type="checkbox"/>	elastic	superuser	<input checked="" type="checkbox"/>
<input type="checkbox"/>	kibana	kibana_system	<input checked="" type="checkbox"/>
<input type="checkbox"/>	logstash_system	logstash_system	<input checked="" type="checkbox"/>

Рис. 8.4

3. Для создания нового пользователя нажмите кнопку **Create User** (Создать пользователя) и добавьте нужную информацию, как показано на рис. 8.5.

4. Нажмите кнопку **Save** (Сохранить).

Теперь, когда пользователь создан, попробуем получить доступ к некоторым REST API Elasticsearch с данными авторизации нового пользователя и посмотрим, что произойдет. Выполните следующую команду и проверьте полученный ответ. Несмотря на успешную аутентификацию, у пользователя нет назначенных ролей. Он получает HTTP-статус 403, говорящий о том, что пользователь не авторизован для выполнения операции:

```
D:\packt\book>curl -s -H "Content-Type: application/json" -u user1:password -XGET http://localhost:9200
```

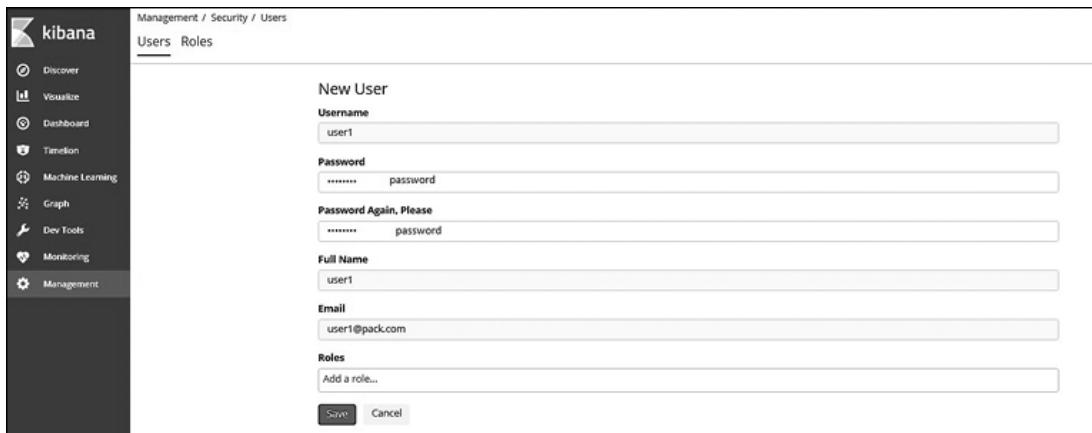


Рис. 8.5

Ответ:

```
{"error": {"root_cause": [{"type": "security_exception", "reason": "action [cluster:monitor/main] is unauthorized for user [user1]"}], "type": "security_exception", "reason": "action [cluster:monitor/main] is unauthorized for user [user1]"}, "status": 403}
```

Аналогичным образом создайте еще одного пользователя с названием **user2**, как показано на рис. 8.6.

The screenshot shows the 'Management / Security / Users' section of the Kibana interface. On the left is a sidebar with various icons for Discover, Visualize, Dashboard, Timeline, Machine Learning, Graph, Dev Tools, Monitoring, and Management. The 'Management' icon is highlighted. The main area is titled 'Users Roles' and contains a 'New User' form. The fields are as follows:

- Username: user2
- Password: password
- Password Again, Please: password
- Full Name: user2
- Email: user2@pack.com
- Roles: Add a role...

At the bottom are 'Save' and 'Cancel' buttons.

Рис. 8.6

Удаление пользователя

Для удаления роли перейдите в раздел пользовательского интерфейса **Users** (Пользователи), выберите созданных пользователей и нажмите кнопку **Delete** (Удалить). Встроенных пользователей удалить невозможно (рис. 8.7).

The screenshot shows the 'Management / Security / Users' section of the Kibana interface. The sidebar and title are the same as in Figure 8.6. The main area displays a table of users:

Full Name	Username	Roles	Reserved
	elastic	superuser	✓
	kibana	kibana_system	✓
	logstash_system	logstash_system	✓
	user1	user1	
<input checked="" type="checkbox"/>	user2	user2	

Below the table, it says '1 users selected'. At the top right of the table area is a 'Delete' button.

Рис. 8.7

Изменение пароля

Перейдите в раздел пользовательского интерфейса **Users** (Пользователи) и выберите пользователя, для которого необходимо изменить пароль. Вы попадете на страницу **User Details** (Сведения о пользователе). Здесь можно отредактировать информацию о пользователе, изменить пароль или удалить пользователя из раздела. Для изменения пароля щелкните на ссылке **Change Password** (Изменить пароль) и введите новый пароль (рис. 8.8). Нажмите кнопку **Save** (Сохранить).

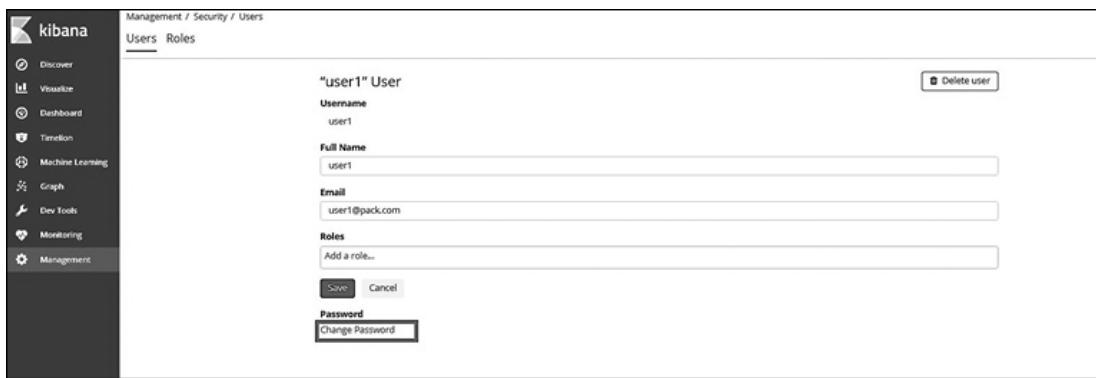


Рис. 8.8



Пароль должен состоять из не менее чем шести символов.

Создание новой роли

Для создания новой роли пользователя перейдите в раздел пользовательского интерфейса **Management** (Управление) и выберите **Roles** (Роли) в разделе **Security** (Безопасность). Если вы находитесь на странице **Users** (Пользователи), перейдите на вкладку **Roles** (Роли). На ней будут перечислены все доступные роли (рис. 8.9). По умолчанию будут показаны встроенные/

резервные роли, которые являются частью области безопасности native в X-Pack.

The screenshot shows the Kibana Management / Security interface with the 'Roles' tab selected. On the left is a sidebar with links like Discover, Visualize, Dashboard, Timeline, Machine Learning, Graph, Dev Tools, Monitoring, and Management. The main area has a search bar and a 'Create role' button. A table lists the following roles:

Role	Status
Role 1	✓ Reserved
ingest_admin	✓
kibana_dashboard_only_user	✓
kibana_system	✓
kibana_user	✓
logstash_admin	✓
logstash_system	✓
machine_learning_admin	✓
machine_learning_user	✓
monitoring_user	✓
remote_monitoring_agent	✓
reporting_user	✓
superuser	✓
transport_client	✓
watcher_admin	✓
watcher_user	✓

Рис. 8.9

Модуль безопасности X-Pack также предоставляет набор встроенных ролей, которые можно назначить пользователям. Эти роли зарезервированы, и ассоциированные с ними привилегии невозможно обновить. Рассмотрим некоторые из встроенных ролей.

- **kibana_system** — разрешает необходимый доступ для чтения и записи в индексы Kibana, управления шаблонами индексов и проверки доступности кластера Elasticsearch. Эта роль также предоставляет доступ к индексам мониторинга (.monitoring-*) и доступ чтения/записи к отчетности (.reporting-*). Эти привилегии по умолчанию есть у пользователя **kibana**.
- **superuser** — предоставляет доступ для выполнения всех операций над кластерами, индексами и данными. Эта роль также позволяет создавать/редактировать пользователей и

роли. Эти привилегии по умолчанию есть у пользователя **elastic**.

- **ingest_admin** – предоставляет разрешения на управление всеми конфигурациями контейнеров и шаблонами индексов.



Полный список всех встроенных ролей и их описание доступны по ссылке <https://www.elastic.co/guide/en/x-pack/master/built-inroles.html>.

Пользователь с ролью **superuser** может создавать другие роли и назначать их другим пользователям в пользовательском интерфейсе Kibana.

Создадим новую роль с привилегией кластера **monitor** и назначим ее пользователю **user1**, чтобы он мог выполнять операции только чтения, такие как запрос сведений о работоспособности кластера, состоянии кластера, информации об узлах, статистики узлов и пр.

Нажмите кнопку **Create Role** (Создать роль) на странице/вкладке **Roles** (Роли) и задайте настройки, как показано на рис. 8.10.

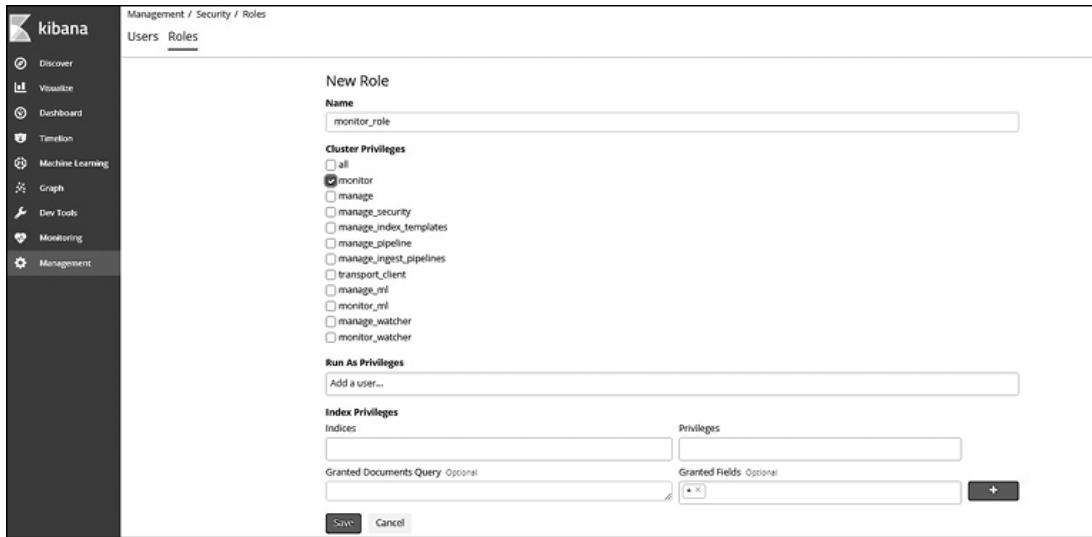


Рис. 8.10

Для назначения только что созданной роли пользователю **user1** перейдите на вкладку **Users** (Пользователи) и выберите **user1**. На странице **User Details** (Сведения о пользователе) в раскрывающемся списке выберите роль **monitor_role** и нажмите кнопку **Save** (Сохранить) (рис. 8.11).



Одному пользователю можно назначить несколько ролей.



Рис. 8.11

Теперь проверим, действительно ли **user1** может получить доступ к некоторым API кластера/узла:

```
curl -u user1:password "http://localhost:9200/_cluster/health?pretty"
{
  "cluster_name" : "elasticsearch",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 53,
  "active_shards" : 53,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 52,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 50.476190476190474
}
```

Выполним ту же команду, как и при создании **user1**, но без назначения ему каких-либо ролей, и посмотрим на разницу:

```
curl -u user1:password "http://localhost:9200"
{
  "name" : "fwDdHSI",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "08wSPsjSQCmeRaxF4iHizw",
```

```
"version" : {  
    "number" : "6.0.0",  
    "build_hash" : "8f0685b",  
    "build_date" : "2017-11-10T18:41:22.859Z",  
    "build_snapshot" : false,  
    "lucene_version" : "7.0.1",  
    "minimum_wire_compatibility_version" :  
    "5.6.0",  
    "minimum_index_compatibility_version" :  
    "5.0.0"  
},  
    "tagline" : "You Know, for Search"  
}
```

Как удалить/редактировать роль. Для удаления роли перейдите на страницу/вкладку **Roles** (Роли), выберите созданные вами роли и нажмите кнопку **Delete** (Удалить). Встроенные роли удалить невозможно (рис. 8.12).

Для редактирования роли перейдите на страницу/вкладку **Roles** (Роли), выберите созданные вами роли, которые нуждаются в редактировании. Пользователь направляется на страницу **Roles Details** (Сведения о роли). Внесите необходимые изменения в разделе привилегий и нажмите кнопку **Save** (Сохранить). На этой странице также можно удалить роль (рис. 8.13).

The screenshot shows the Kibana Management interface under the Security section, specifically the Roles tab. On the left is a dark sidebar with various navigation options. The main area has a header 'Management / Security' and 'Users Roles'. Below is a table with a search bar at the top. The table lists roles with checkboxes and checkmarks indicating their status. A 'Delete' button is located in the top right corner of the table.

<input type="checkbox"/> Role	Reserved	<input type="button" value="Delete"/>
<input type="checkbox"/> ingest_admin	✓	
<input type="checkbox"/> kibana_dashboard_only_user	✓	
<input type="checkbox"/> kibana_system	✓	
<input type="checkbox"/> kibana_user	✓	
<input type="checkbox"/> logstash_admin	✓	
<input type="checkbox"/> logstash_system	✓	
<input type="checkbox"/> machine_learning_admin	✓	
<input type="checkbox"/> machine_learning_user	✓	
<input checked="" type="checkbox"/> monitor_role		
<input type="checkbox"/> monitoring_user	✓	
<input type="checkbox"/> remote_monitoring_agent	✓	

Рис. 8.12

The screenshot shows the Kibana Management interface under the Security section, specifically the Roles tab. On the left is a dark sidebar with various navigation options. The main area has a header 'Management / Security / Roles' and 'Users Roles'. Below is a form for creating a new role named 'monitor_role'. It includes sections for 'Cluster Privileges' (with 'monitor' and 'manage' selected), 'Run As Privileges' (with 'Add a user...'), 'Index Privileges' (with 'Indices' and 'Privileges' fields), and 'Granted Documents Query' and 'Granted Fields' sections. At the bottom are 'Save' and 'Cancel' buttons.

Рис. 8.13

Безопасность уровня документов или уровня полей

Теперь, когда вы знаете, как создавать новых пользователей, роли и назначать роли пользователям, рассмотрим возможности безопасности для документов и полей в определенных индексах/документах.

В примере данных, который мы импортировали в начале главы, есть два индекса: сотрудников (`employee`) и отделов (`department`).

Пример 1. Когда пользователь ищет информацию о сотруднике, он не должен получать в результатах поиска сведения о зарплате и адресе из документов в индексе `employee`.

В этом случае поможет безопасность уровня поля. Создадим новую роль (`employee_read`) с привилегией `read` для индекса `employee`. Для ограничения полей в разделе **Granted Fields** (Разрешенные поля) выберите те поля, к которым можно разрешить доступ пользователю, как показано на рис. 8.14.

The screenshot shows the Kibana Management interface under the Security / Roles tab. A new role named 'employee_read' is being created. In the 'Granted Fields' section, several fields are selected: gender, state, email, address1.keyword, _field_names, address2.keyword, name, state.keyword, _source, _id, and _uid. Other fields like _score and _type are present but not selected.

Рис. 8.14



При создании роли можно указать один набор привилегий для нескольких индексов. Для этого необходимо добавить один или несколько индексов в поле Indices (Индексы) или же указать разные привилегии для разных индексов, нажав кнопку + в разделе Index Privileges (Привилегии индексов).

Назначьте пользователю **user2** новую роль (рис. 8.15).

The screenshot shows the Kibana management interface under the 'Management / Security / Users' section. On the left is a sidebar with various navigation options like Discover, Visualize, Dashboard, Timeline, Machine Learning, Graph, Dev Tools, Monitoring, and Management. The 'Management' option is currently selected. In the main area, there's a sub-menu for 'Users Roles'. A specific user profile for 'user2' is displayed, showing the following fields:

- Username: user2
- Full Name: user2
- Email: user2@pack.com
- Roles: A dropdown menu titled 'Add a role...' lists several roles: reporting_user, kibana_system, logstash_admin, transport_client, superuser, ingest_admin, monitor_role, and employee_read. The 'employee_read' role is highlighted with a black rectangle.

A small 'Delete user' button is located in the top right corner of the user profile area.

Рис. 8.15

Теперь запустите поиск по индексу сотрудников и посмотрите, какие поля будут в ответе. Как видно в ответе ниже, мы успешно ограничили пользователей в получении доступа к информации о зарплате и адресе:

```
curl -u user2:password "http://localhost:9200/employee/_search?pretty"
{
  "took" : 20,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 3,
    "max_score" : 1.0,
    "hits" : [
      {
        "id" : "1",
        "name" : "John Doe",
        "age" : 30,
        "city" : "New York",
        "salary" : 5000
      },
      {
        "id" : "2",
        "name" : "Jane Doe",
        "age" : 28,
        "city" : "Los Angeles",
        "salary" : 4500
      },
      {
        "id" : "3",
        "name" : "Mike Johnson",
        "age" : 32,
        "city" : "Chicago",
        "salary" : 5500
      }
    ]
}
```



```

        "name" : "user1",
        "state" : "NE",
        "email" : "user1@packt.com"
    }
}
]
}
}

```

Пример 2. Требуется создать многопользовательский индекс и ограничить доступ определенных пользователей к отдельным документам. Например, сделать так, чтобы пользователь **user1** мог искать по индексу отдела и получать только те документы, которые относятся к IT-отделу.

Создайте роль `department_IT_role` и предоставьте привилегию `read` для индекса `department`. Для ограничения документов укажите запрос в разделе **Granted Documents Query** (Разрешенные запросы документов). Запрос должен быть в формате Elasticsearch Query DSL (рис. 8.16).



Рис. 8.16

Свяжите созданную роль с пользователем **user1** (рис. 8.17).



Рис. 8.17

Теперь нужно проверить работоспособность этих настроек, выполнив поиск по индексу `department` с данными аутентификации пользователя `user1`:

```
curl -u user1:password "http://localhost:9200/department/_search?pretty"
{
  "took" : 1,
  "timed_out" : false,
  "_shards" : {
    "total" : 5,
    "successful" : 5,
    "skipped" : 0,
    "failed" : 0
  },
  "hits" : {
    "total" : 1,
    "max_score" : 1.0,
    "hits" : [
      {
        "_index" : "department",
        "_type" : "department",
```

```
        "_id" : "3wuULGABsx353N7xt4k6",
        "_score" : 1.0,
        "_source" : {
            "name" : "IT",
            "employees" : 50
        }
    }
]
}
}
```

API безопасности X-Pack

В предыдущем разделе вы узнали, как управлять пользователями и ролями в пользовательском интерфейсе Kibana. Однако при многократном выполнении этих операций хотелось бы делать это программно из приложений. Вот где могут пригодиться API безопасности X-Pack. Они состоят из REST API, которые можно применять для управления пользователями/ролями, назначения ролей пользователям, проведения аутентификаций и проверки наличия необходимых привилегий у аутентифицированного пользователя. Эти API проводят операции в области безопасности `native`. Пользовательский интерфейс Kibana задействует эти API на внутреннем уровне для управления пользователями/ролями. Для выполнения этих API пользователь должен иметь привилегии `superuser` или `manage_security`.

API управления пользователями

Предоставляет набор API для создания, редактирования или удаления пользователей из области безопасности `native`.

Список доступных API:

GET /_xpack/security/user	Список всех пользователей
GET /_xpack/security/user/<username>	Выводит сведения о конкретном пользователе
DELETE /_xpack/security/user/<username>	Удаляет пользователя
POST /_xpack/security/user/<username>	Создает нового пользователя
PUT /_xpack/security/user/<username>	Обновляет информацию о существующем пользователе
PUT /_xpack/security/user/<username>/_disable	Блокирует существующего пользователя
PUT /_xpack/security/user/<username>/_enable	Активизирует заблокированного пользователя
PUT /_xpack/security/user/<username>/_password	Изменяет пароль

Параметр `username` в указанном пути определяет пользователя для выполняемой операции. Тело запроса принимает такие параметры, как `email`, `full_name` и `password` в качестве строки, а `roles` — в качестве списка.

Пример 1. Создадим нового пользователя `user3` с назначенной ему ролью `monitor_role`:

```
curl -u elastic:elastic -X POST
http://localhost:9200/_xpack/security/user/user3 -H 'content-type:
application/json' -d '
{
  "password" : "randompassword",
  "roles" : [ "monitor_role"],
  "full_name" : "user3",
  "email" : "user3@packt.com"
}'
```

Ответ:

```
user": {"created": true}}
```

Пример 2. Получим список всех пользователей:

```
curl -u elastic:elastic -XGET
```

```
http://localhost:9200/_xpack/security/user?  
pretty
```

Пример 3. Удалим пользователя **user3**:

```
curl -u elastic:elastic -XDELETE  
http://localhost:9200/_xpack/security/user/user  
3
```

Ответ:

```
{"found":true}
```

Пример 4. Изменим пароль:

```
curl -u elastic:elastic -XPUT  
http://localhost:9200/_xpack/security/user/user  
2/_password -H "contenttype:  
application/json" -d "{\"password\":  
\"newpassword\"}"
```



Обратите внимание, что при использовании команд curl на устройствах с Windows они не будут срабатывать при наличии одинарных кавычек в тексте (''). В предыдущем примере показано применение команды curl на устройстве под управлением Windows. Убедитесь также, что вы не используете двойные кавычки в теле команды, как это показано в предыдущем примере.

API управления ролями

Предоставляет набор API для создания, редактирования, удаления и получения ролей из области безопасности native.

Список доступных API:

GET /_xpack/security/role	Выводит список всех ролей
GET /_xpack/security/role/<rolename>	Выводит информацию о конкретной роли
POST /_xpack/security/role/<rolename>/_clear_cache	Исключает роль из кэша ролей native
POST /_xpack/security/role/<rolename>	Создает роль
PUT /_xpack/security/role/<rolename>	Обновляет существующую роль

Параметр `rolename` в указанном пути определяет роль для выполняемой операции. Тело запроса содержит следующие параметры: `cluster`, который принимает список привилегий кластера; `indices` принимает список объектов, указывающих привилегии индекса; `run_as` содержит список пользователей, от имени которых могут действовать обладатели этой роли.

Параметр `indices` содержит объект с такими параметрами, как `names`, принимающий список названий индексов; `field_security`, принимающий список полей, для которых необходимо предоставить доступ чтения; `privileges`, принимающий список привилегий индекса; `query`, принимающий запрос для фильтрации документов.

Пример 1. Создадим новую роль с безопасностью на уровне поля для работы от имени индекса сотрудников:

```
curl -u elastic:elastic -X POST
http://localhost:9200/_xpack/security/role/employee_read_new -H
'contenttype: application/json' -d '{
  "indices": [
    {
      "names": [ "employee" ],
      "privileges": [ "read" ],
    }
  ]
}'
```

```
        "field_security" : {
            "grant" : [ "*" ],
            "except": [ "address*","salary" ]
        }
    }
]
}'
```

Ответ:

```
role":{"created":true}}
```



В отличие от пользовательского интерфейса Kibana, в котором не предусмотрено возможностей для исключения полей из пользовательского доступа, вы легко можете включать или исключать поля с помощью API безопасности в рамках безопасности на уровне поля. В предыдущем примере мы ограничили доступ к полю salary и любым полям, которые начинаются с текста/строки address.

Пример 2. Получим сведения об определенной роли:

```
curl -u elastic:elastic -XGET
http://localhost:9200/_xpack/security/role/employee_read_new?pretty
```

Ответ:

```
{
  "employee_read" : {
```

```
"cluster" : [ ],
"indices" : [
  {
    "names" : [
      "employee"
    ],
    "privileges" : [
      "read"
    ],
    "field_security" : {
      "grant" : [
        "*"
      ],
      "except" : [
        "address*",
        "salary"
      ]
    }
  }
],
"run_as" : [ ],
"metadata" : { },
"transient_metadata" : {
  "enabled" : true
}
}
```

Пример 3. Удалим роль:

```
curl -u elastic:elastic -XDELETE
```

`http://localhost:9200/_xpack/security/role/employee_read`

Ответ:

`{"found":true}`



По аналогии с API управления пользователем и ролями вы можете применять API назначения пользователей для присвоения ролей пользователям. Более детально об этом API вы можете узнать по ссылке <https://www.elastic.co/guide/en/elasticsearch/reference/master/security-api-role-mapping.html>.

Мониторинг Elasticsearch

Для мониторинга на уровнях кластера, узла или индекса в Elasticsearch предусмотрен широкий набор API, известных как *API статистики*. Некоторые из этих API: `_cluster/stats`, `_nodes/stats`, `myindex/stats`. Они предоставляют информацию о состоянии и мониторинге в реальном времени, а статистика выдается в форматах `point-in-time` и `.json`. Если вы администратор или разработчик, возможно, вы будете заинтересованы в статистике в реальном режиме времени и в накопленной статистике о работе в Elasticsearch. Такие инструменты помогают понимать и анализировать поведение (состояние или производительность) кластера. Именно на этом этапе пригодится функционал мониторинга X-Pack.

Компоненты мониторинга X-Pack позволяют вам легко отслеживать работу составляющих Elastic Stack (Elasticsearch,

Kibana и Logstash) из Kibana. X-Pack состоит из агентов мониторинга, которые работают в каждом процессе (Elasticsearch, Kibana и Logstash) и периодически собирают и индексируют метрики о состоянии и производительности. Далее вы можете выполнить визуализацию этих данных в разделе **Monitoring** (Мониторинг) UI Kibana. Вы можете выбрать необходимые панели управления среди предустановленных в Kibana и с их помощью легко визуализировать и анализировать свои данные в реальном времени.

По умолчанию все собранные в X-Pack метрики индексируются в пределах того же кластера, который вы мониторите. Однако в работе крайне рекомендуется иметь отдельный кластер для хранения этих метрик (рис. 8.18). Такое решение имеет следующие преимущества.

- Возможность мониторинга нескольких кластеров из централизованного места.
- Уменьшение нагрузки и экономия дискового пространства рабочих кластеров, поскольку метрики хранятся в отдельном кластере.
- Доступ к данным мониторинга есть всегда, даже при выходе из строя отдельных кластеров.
- Вы можете установить разные меры безопасности для кластера мониторинга и рабочих кластеров.

Как уже говорилось ранее, все собранные в X-Pack метрики индексируются в пределах того же кластера, который вы мониторите. Если выбран отдельный кластер для мониторинга, необходимо настроить доставку метрик соответствующим

образом. Это можно сделать в файле `elasticsearch.yml` каждого узла, как показано в коде ниже:

```
xpack.monitoring.exporters:  
  id1:  
    type: http  
    host:  
      ["http://dedicated_monitoring_cluster:port"]
```

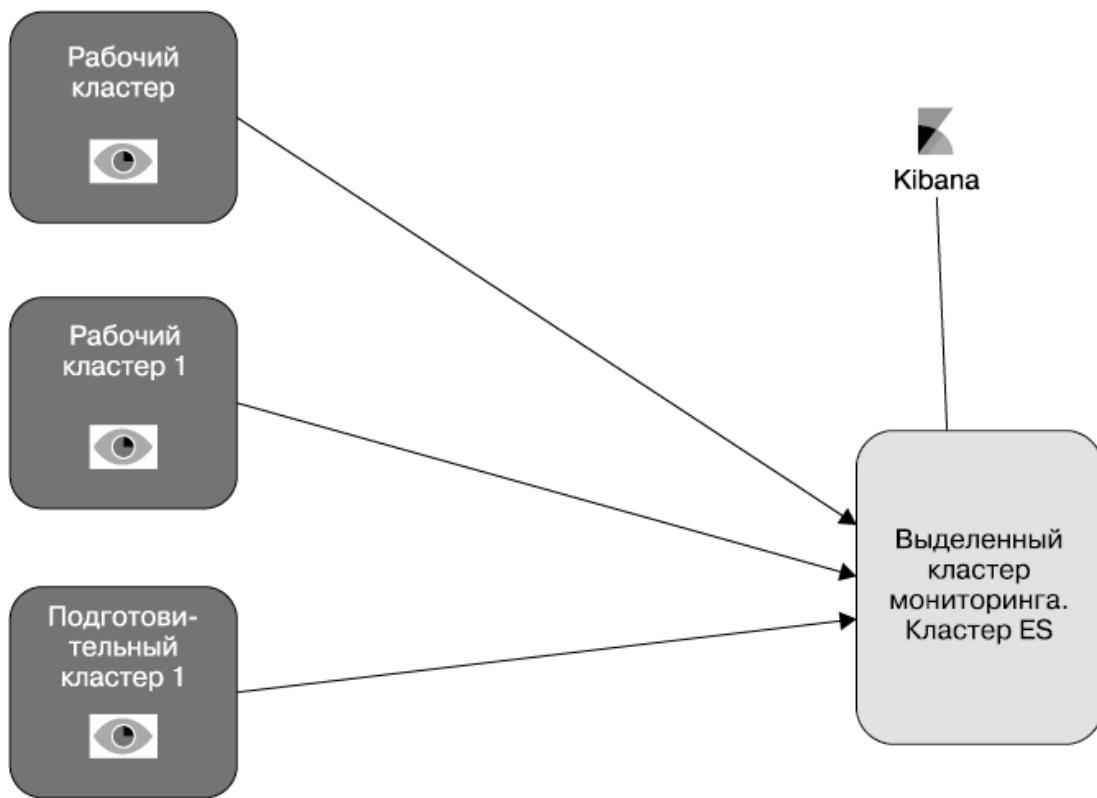


Рис. 8.18



Не обязательно устанавливать X-Pack на выделенный кластер мониторинга, однако рекомендуется установить его и там. Если вы выберете такой вариант конфигурации, убедитесь, что во

время настройки указали данные авторизации пользователя (auth.username и auth.password). Метрики сохраняются в индексе на уровне системы; он имеет шаблон индекса .monitoring-*.

Раздел Monitoring (Мониторинг) пользовательского интерфейса

Для доступа к настройкам мониторинга зайдите в Kibana и перейдите в раздел **Monitoring** (Мониторинг) на боковой панели навигации (рис. 8.19).

На этой странице вы можете увидеть все метрики, доступные для Elasticsearch и Kibana. Более подробную информацию вы можете получить, нажимая кнопки **Overview** (Обзор), **Nodes** (Узлы), **Indices** (Индексы) или **Instances** (Процессы). Все метрики, которые вы видите на этой странице, автоматически обновляются каждые 10 с, и по умолчанию разрешено посмотреть данные за 1 ч. Эти настройки можно изменить в разделе **Time Filter** (Фильтр времени), который находится слева вверху страницы. Отображается также название кластера, в данном случае — **elasticsearch**.

The screenshot shows the Kibana Monitoring interface. On the left, a sidebar menu includes options like Discover, Visualize, Dashboard, Timeline, Machine Learning, Graph, Dev Tools, Monitoring (which is selected and highlighted in blue), and Management. Below that are links for elastic (Logout) and Collapse. The main content area has a header 'Clusters' and 'elasticsearch'. It displays a trial license expiration notice ('Your Trial license will expire on January 2, 2018.') and 'Top Cluster Alerts' with two entries: 'Elasticsearch cluster status is yellow. Allocate missing replica shards.' (last checked Dec 3, 2017) and 'Configuring TLS will be required to apply a Gold or Platinum license when security is enabled. See documentation for details.' (last checked Dec 3, 2017). Below this is a 'View all alerts' section. The 'Elasticsearch' section shows an overview with 'Nodes: 1', 'Disk Available: 124GB / 262GB (47.25%)', 'JVM Heap: 30.46% (302MB / 991MB)', 'Indices: 13', 'Documents: 11,196', 'Disk Usage: 6MB', 'Primary Shards: 21', and 'Replica Shards: 0'. The 'Health' status is marked as 'Yellow'. The 'Kibana' section shows an overview with 'Instances: 1', 'Connections: 199', and 'Memory Usage: 10.03% (144MB / 1GB)'. The 'Health' status is marked as 'Green'.

Рис. 8.19



По умолчанию агент отправляет данные о наблюдаемых процессах каждые 10 с. Вы можете изменить эту настройку в файле (`elasticsearch.yml`), отредактировав параметр `pack.monitoring.collection.interval` соответствующим образом.

Метрики Elasticsearch

Вы можете мониторить данные производительности Elasticsearch на уровне кластера, узла и индекса. Страница **Monitoring** (Мониторинг) Elasticsearch содержит три вкладки, каждая отображает метрики на уровне кластера, узла и индекса. Эти вкладки называются **Overview** (Обзор), **Nodes** (Узлы), **Indices** (Индексы) соответственно. Для перехода к пользовательскому интерфейсу мониторинга Elasticsearch щелкните на одноименной ссылке.

Вкладка Overview (Обзор)

Метрики на уровне кластера предоставляют информацию, собранную из всех узлов. Это первое место, с которого начинается мониторинг кластера Elasticsearch. Просмотреть метрики на уровне кластера можно, перейдя на вкладку **Overview** (Обзор) или щелкнув на ссылке **Overview** (Обзор). На вкладке выводятся все ключевые метрики, по которым можно судить об общем состоянии кластера Elasticsearch (рис. 8.20).

В состав ключевых метрик входят статус кластера, количество узлов и индексов, использованная память, количество шардов, количество неназначенных шардов, количество документов в индексе, дисковое пространство,

использованное для хранения документов, время работы и версия Elasticsearch. На вкладке **Overview** (Обзор) также отображаются диаграммы, которые показывают производительность по поиску и индексированию, а таблица внизу содержит информацию о каких-либо шардах, проходящих восстановление.

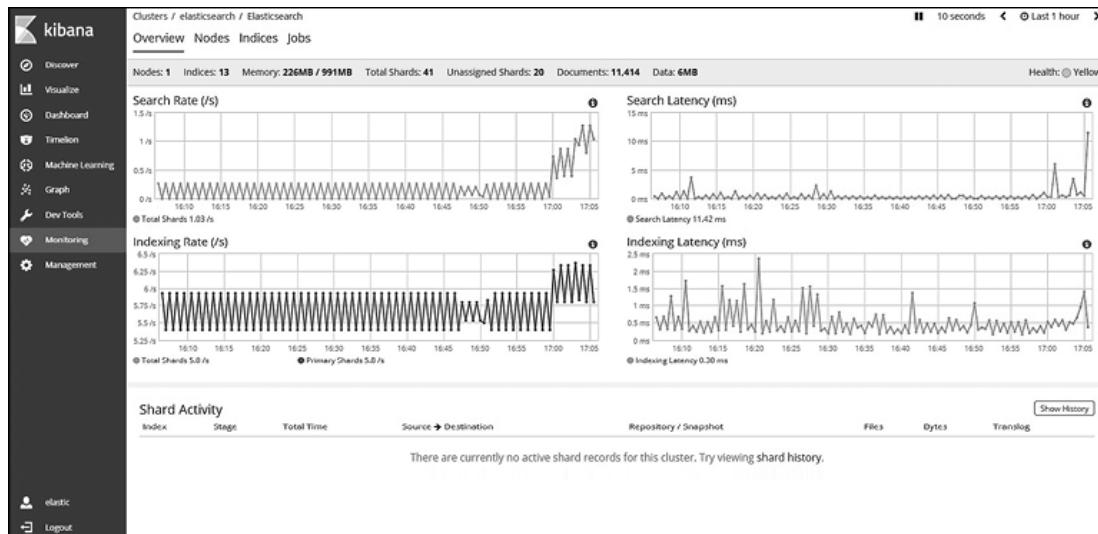


Рис. 8.20



При щелчке на информационном значке в верхнем правом углу каждой диаграммы можно получить описание метрики.

На данной вкладке метрики собираются только на уровне кластера; следовательно, при мониторинге работы Elasticsearch вы можете упустить важные параметры, которые влияют на общее состояние кластера. Например, метрика **Memory Used** (Использованная память) показывает средний расход памяти, использованной во всех узлах. Однако один из узлов может использовать всю память, а другие — почти не использовать ее.

Следовательно, с точки зрения администрирования необходимо производить мониторинг и на уровне узла.

Вкладка **Nodes** (Узлы)

Перейдя на вкладку **Nodes** (Узлы), вы увидите общую информацию о каждом узле кластера (рис. 8.21).

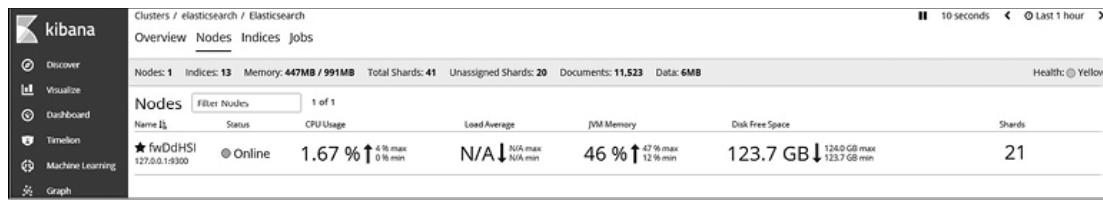


Рис. 8.21

Для каждого узла доступна следующая информация: название узла, статус, нагрузка ЦП (среднее, минимальное, максимальное значения), средняя нагрузка (среднее, минимальное, максимальное значения), память JVM (среднее, минимальное, максимальное значения), свободное дисковое пространство (среднее, минимальное, максимальное значения) и количество назначенных шардов. Вы также можете узнать, является ли узел главным (обозначен звездочкой), и получить информацию о хосте доставки и порте.

Щелкнув на названии узла, вы увидите информацию об узле. Подробные сведения об узле разделены на две вкладки: **Overview** (Обзор) и **Advanced** (Дополнительно). Обзор узла выглядит следующим образом (рис. 8.22).

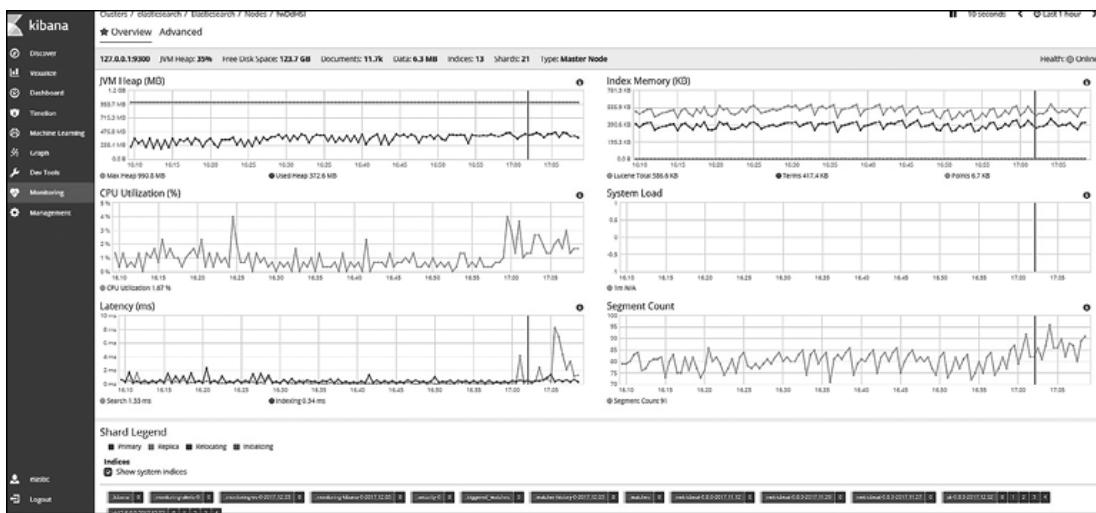


Рис. 8.22

В верхней области вкладки **Overview** (Обзор) узла представлена следующая информация: статус узла, транспортный IP-адрес узла, процентное использование кучи JVM, доступное свободное дисковое пространство, общее количество документов в узле (включая документы в основных шардах и их репликах), общий объем использованного дискового пространства, количество индексов в узле, количество шардов и тип узла (главный узел, узел данных, узел поглощения и узел координации).

Здесь вы также наглядно увидите, в каком объеме используются куча JVM, память индекса, сведения о коэффициенте загрузки ЦП средней нагрузки на систему, задержке (в миллисекундах) и количестве сегментов. В разделе **Shard Legend** (Легенда шардов) вы можете просмотреть статусы шардов различных индексов.



Вы можете увидеть статус шардов всех индексов, созданных X-Pack, если установите флажок **Show system indices** (Показывать

системные индексы).

На вкладке **Advanced** (Дополнительно) узла вы найдете визуализации других метрик, таких как подсчет и длительность сборки мусора (*GC*), подробный расход памяти на уровнях Lucene и Elasticsearch, время индексирования (в миллисекундах), скорость запросов, индексирования, ветки чтения и статистика Cgroup (рис. 8.23).

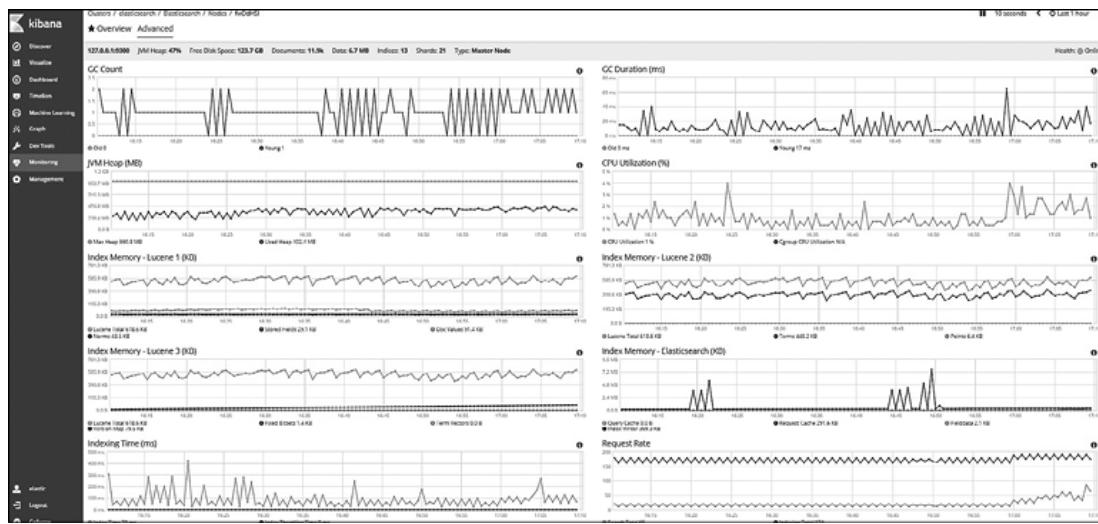


Рис. 8.23

Вкладка Indices (Индексы)

Перейдя на вкладку **Indices** (Индексы), вы увидите подробности о каждом индексе кластера (рис. 8.24).

The screenshot shows the Kibana interface for the 'Indices' section. On the left is a dark sidebar with various monitoring and management tools. The main area displays a table of indices. At the top of the table, it says 'Nodes: 1 Indices: 13 Memory: 432MB / 991MB Total Shards: 41 Unassigned Shards: 20 Documents: 12,644 Data: 7MB'. The table has columns for Name, Status, Document Count, Data, Index Rate, Search Rate, and Unassigned Shards. The data includes:

Name	Status	Document Count	Data	Index Rate	Search Rate	Unassigned Shards
metricbeat-6.0.0-2017.11.12	Yellow	0	264.0 B	0 /s	0 /s	1
.monitoring-kibana-6-2017.12.03	Yellow	70	100.6 KB	0.1 /s	0.2 /s	1
.kibana	Yellow	108	290.3 KB	0 /s	0.4 /s	1
sk12-6.0.0-2017.12.02	Yellow	99	544.4 KB	0 /s	0 /s	5
sk-6.0.0-2017.12.02	Yellow	22	187.6 KB	0 /s	0 /s	5
metricbeat-6.0.0-2017.11.27	Yellow	210	219.3 KB	0 /s	0 /s	1
metricbeat-6.0.0-2017.11.26	Yellow	4.1k	915.3 KB	0 /s	0 /s	1
.monitoring-alerts-6	Yellow	1	8.8 KB	0.03 /s	0.17 /s	1
.watches	Yellow	5	103.1 KB	0.17 /s	0 /s	1
.watcher-history-6-2017.12.03	Yellow	497	741.0 KB	0.17 /s	0 /s	1
.triggered_watches	Yellow	0	16.2 KB	0.17 /s	0 /s	1
.monitoring-es-6-2017.12.03	Yellow	6.9k	3.8 MB	5.7 /s	4.3 /s	1

Рис. 8.24



Вы можете увидеть статус шардов всех индексов, созданных X-Pack, если установите флагок Show system indices (Показывать системные индексы).

Для каждого индекса доступна следующая информация: название индекса, статус индекса, общее количество документов, использованное дисковое пространство, скорость индексирования в секунду, скорость поиска в секунду и количество неназначенных шардов.

Информацию об индексе можно получить, щелкнув на его названии. Подробные сведения об индексе разделены на две вкладки: **Overview** (Обзор) и **Advanced** (Дополнительно). Пример вкладки **Overview** (Обзор) приведен на рис. 8.25.

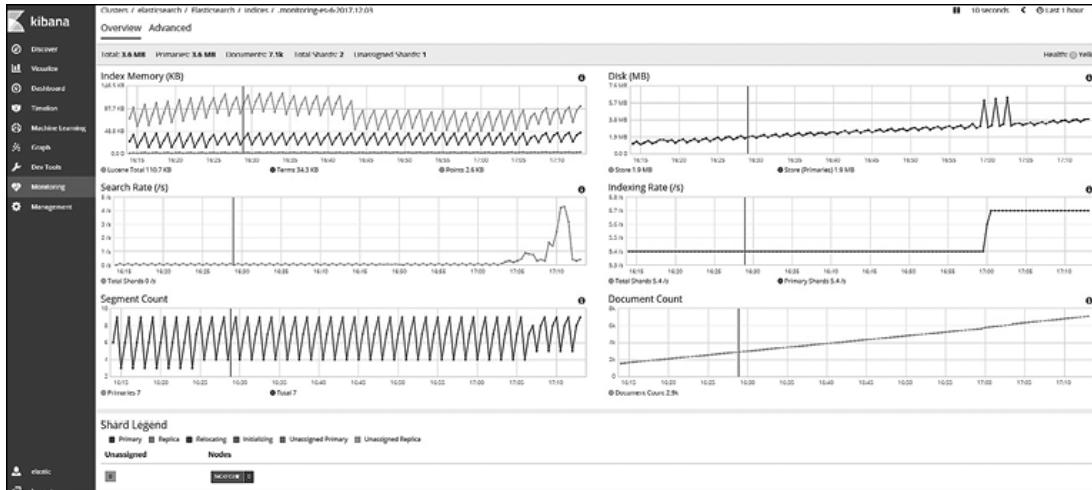


Рис. 8.25

В верхней области вкладки представлена следующая информация: статус индекса, общее количество документов, использованное дисковое пространство, количество шардов (основные + реплики) и неназначенные шарды.

Кроме того, здесь вы можете наглядно увидеть, каков процент использования памяти индекса (в килобайтах), размер индекса (в мегабайтах), скорость поисков в секунду, скорость индексирования в секунду, количество сегментов и количество документов. В области **Shard Legend** (Легенда шардов) вы можете просмотреть статусы шардов, принадлежащих индексу, и информацию об узлах, которым назначены шарды.

На вкладке **Advanced** (Дополнительно) визуализированы другие метрики, такие как расход памяти индекса на уровнях Lucene и Elasticsearch, время индексирования (в миллисекундах), скорость и время запросов, время обновления (в миллисекундах), используемое дисковое пространство и количество сегментов (рис. 8.26).



На стартовой странице пользовательского интерфейса Monitoring (Мониторинг) вы можете аналогичным способом мониторить/визуализировать метрики Kibana. Для этого нужно нажать кнопку Overview (Обзор) или Instances (События) в разделе Kibana.

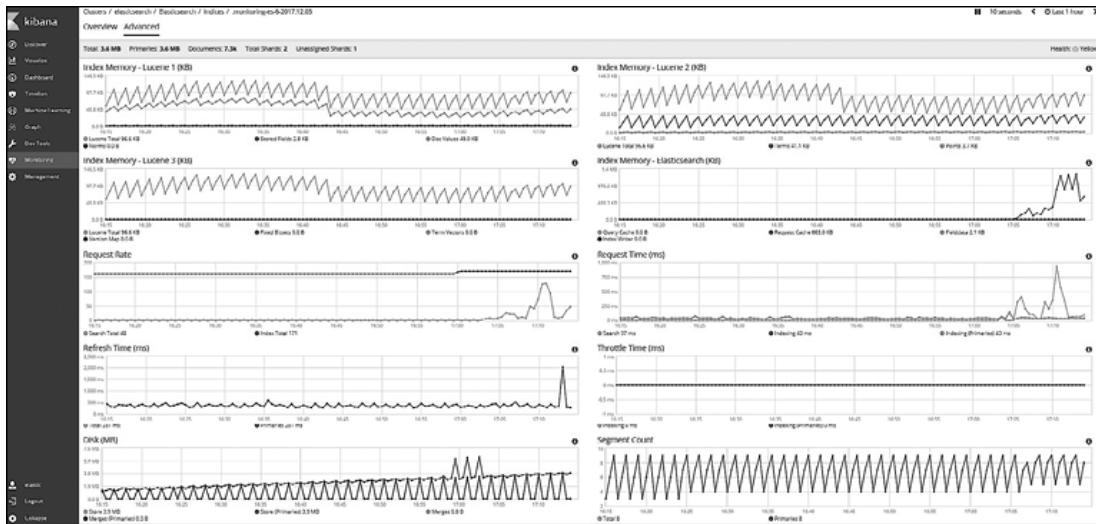


Рис. 8.26

Уведомления

Пользовательский интерфейс Kibana предоставляет красивые визуализации, которые помогают анализировать ваши данные и обнаруживать аномалии в режиме реального времени. Однако администратор или аналитик не могут часами сидеть перед панелями управления в попытке обнаружить аномалию, чтобы оперативно предпринять соответствующие действия. Было бы отлично, если бы администратор получал уведомления, когда происходят определенные события.

- Выходит из строя один из серверов под наблюдением.

- Кластер Elasticsearch переходит в красное/желтое состояние из-за того, что некоторые узлы покидают кластер.
- Использование ресурсов диска или ЦП превышает определенное значение.
- Обнаружено вторжение по сети.
- В логах найдены ошибки.

В таких случаях на помощь приходит компонент уведомления X-Pack. Он называется **Watcher** (*Наблюдатель*) и предоставляет такой функционал, как автоматическое наблюдение за изменениями или аномалиями в данных, хранящихся в Elasticsearch, а также позволяет предпринимать необходимые действия. Уведомления X-Pack включены по умолчанию как часть стандартной установки.

Наблюдатель предоставляет набор REST API для создания и тестирования наблюдателей, а также управления ими. Эти действия можно выполнять и на странице **Watcher** (*Наблюдатель*) в пользовательском интерфейсе Kibana. На внутреннем уровне будут применяться REST API наблюдателей для управления ими.

Структура наблюдателя

Наблюдатель состоит из следующих компонентов (рис. 8.27).

- **schedule** — используется для указания интервала времени для расписания работы наблюдателя.
- **query** — применяется для указания запроса с целью получения данных из Elasticsearch и запуска в качестве

источника ввода. Для уточнения запросов возможно использование Elasticsearch-запросов DSL/Lucene.

- **condition** — используется для указания условий касательно вводных данных, полученных из запроса, при которых следует или не следует предпринимать действия.
- **action** — применяется для указания типа действий, таких как отправка сообщения по электронной почте, отправка уведомления Slack, запись события в определенный лог и др.



Рис. 8.27

Рассмотрим пример наблюдателя и узнаем принцип его действия в деталях. Следующий фрагмент кода создает наблюдателя:

```
curl -u elastic:elastic -X POST
```

```
http://localhost:9200/_xpack/watcher/watch/logs  
tash_error_watch -H  
'content-type: application/json' -d '{  
  
    "trigger" : { "schedule" : { "interval" :  
"30s" }},  
    "input" : {  
        "search" : {  
            "request" : {  
                "indices" : [ "logstash*" ],  
                "body" : {  
                    "query" : {  
                        "match" : { "message": "error" }  
                    }  
                }  
            }  
        }  
    },  
    "condition" : {  
        "compare" : { "ctx.payload.hits.total" : {  
"gt" : 0 }}  
    },  
    "actions" : {  
        "log_error" : {  
            "logging" : {  
                "text" : "The number of errors in logs  
is  
                {{ctx.payload.hits.total}}"  
            }  
        }  
    }  
}
```

}'



Чтобы можно было создать наблюдателя, у пользователя должна быть привилегия `watcher_admin`.

- `trigger` — используется для указания расписания, а именно — как часто должно выполняться наблюдение. Как только наблюдатель создан, его триггер регистрируется движком `scheduler` для корректного запуска наблюдателя.

Можно настроить несколько типов триггеров расписания для указания, когда наблюдатель должен начинать работу. Доступны следующие триггеры расписания: интервал, каждый час, каждый день, каждую неделю, каждый месяц, каждый год и cron.

В предыдущем фрагменте кода триггер был настроен с расписанием 30 с, что означает запуск наблюдателя каждые 30 с.

Пример установки ежечасного триггера. Следующий фрагмент кода показывает, как установить ежечасный триггер, который запускает наблюдатель каждую 45-ю минуту часа:

```
{  
    "trigger" : {  
        "schedule" : {  
            "hourly" : { "minute" : 45 }  
        }  
    }  
}
```

Можно также указать массив минут. Следующий фрагмент кода показывает, как указать ежечасный триггер, который запускает наблюдателя каждую 15-ю и 45-ю минуту часа:

```
{  
    "trigger" : {  
        "schedule" : {  
            "hourly" : { "minute" : [ 15, 45 ] }  
        }  
    }  
}
```

Пример для настройки наблюдателя, срабатывающего ежедневно в 8 ч вечера:

```
{  
    "trigger" : {  
        "schedule" : {  
            "daily" : { "at" : "20:00" }  
        }  
    }  
}
```

Пример для настройки наблюдателя, срабатывающего каждую неделю в 10 ч утра по понедельникам и в 8 ч вечера по пятницам:

```
{  
    "trigger" : {  
        "schedule" : {  
            "weekly" : [  
                { "on" : "monday", "at" : "10:00"  
            },  
                { "on" : "friday", "at" : "10:00" }  
            ]  
        }  
    }  
}
```

```
    }
}
```

Пример настройки расписания с применением синтаксиса cron. Следующий фрагмент кода дает указание наблюдателю срабатывать ежечасно, каждую 45-ю минуту:

```
{
  "trigger" : {
    "schedule" : {
      "cron" : "0 45 * * * ?"
    }
  }
}
```

- `input` — применяется для указания источника ввода для загрузки данных в контекст выполнения наблюдателя. Данные фигурируют как *нагрузка наблюдателя (watcher payload)* и будут доступны на последующих стадиях работы наблюдателя. Их можно будет использовать для создания условий или выполнения над ними действий. Вы можете получить доступ к нагрузке, используя переменную `ctx.payload.*`:

```
"input" : {
  "search" : {
    "request" : {
      "indices" : [ "logstash*" ],
      "body" : {
        "query" : {
          "match" : { "message": "error" }
        }
      }
    }
}
```

```
        }
    }
}
```

Как видно из предыдущего фрагмента кода, ввод типа `search` используется для указания запроса, который должен быть выполнен в Elasticsearch для присвоения данным статуса нагрузки наблюдателя. Запрос выбирает все документы в индексах шаблона `logstash*`, которые содержат `error` в поле `message`.



Вы также можете указывать в разделе `input` следующие типы ввода: `simple` для загрузки статичных данных, `http` для загрузки ответов `http` и `chain` для предоставления серии вводов.

- `condition` — применяется для указания, условия в зависимости от нагрузки, для определения того, необходимо выполнять действие или нет:

```
"condition" : {
    "compare" : { "ctx.payload.hits.total" :
    { "gt" : 0 }}}
}
```

Как видно в предыдущем фрагменте кода, используется тип `compare` для определения наличия документов в нагрузке; если они есть, тогда будет выполнено действие.

Условие типа `compare` применяется для указания простых сравнений типа `eq`, `noteq`, `gt`, `gte`, `lt` и `lte` к значению в нагрузке наблюдателя.



Поддерживаются также следующие условия: always, которое всегда устанавливает условие наблюдателя в true; never, которое всегда устанавливает условие в false; array_compare выполняет сравнение с массивом значений, чтобы определить условия наблюдателя; script создает сценарий для определения условия наблюдателя.

- `actions` — применяется для указания одного или нескольких действий, которые необходимо предпринять, если условие наблюдателя становится равным `true`:

```
"actions" : {
    "log_error" : {
        "logging" : {
            "text" : "The number of errors in
logs is
{{ctx.payload.hits.total}}"
        }
    }
}
```

Как видно в предыдущем фрагменте кода, используется действие записи определенного текста в лог при выбранном состоянии наблюдателя. Запись выполняется в логи Elasticsearch. Количество ошибок вычисляется динамически с помощью поля (`hits.total`) в нагрузке. Для доступа к нагрузке предназначена переменная `ctx.payload.*`.



Наблюдатель поддерживает следующие типы действий: email, webhook, index, logging, hipchat, Slack и pagerduty.

Во время работы наблюдателя, как только условие соответствует требованиям, выполняется решение по настроенному действию: следует прервать или продолжить его. Основное назначение прерывания действия состоит в предотвращении слишком частого выполнения одного и того же действия для одного наблюдателя.

Наблюдатель поддерживает два типа прерывания.

- **Прерывание на основе времени.** Вы можете указать период прерывания с помощью параметра `throttle_period` как часть конфигурации действий или ограничить на уровне наблюдателя (касательно всех действий) частоту выполнения действия. Глобальный период прерывания по умолчанию — 5 с.
- **Прерывание на основе ACK.** Используя ACK Watch API, можно предотвращать повторное выполнение действия наблюдателя, пока условием наблюдения является `true`.

Наблюдатели хранятся в специальном индексе с названием `.watches`. Каждый раз, когда выполняется работа наблюдателя, в индексе истории наблюдателя (под названием `.watches-history-6-*`) сохраняется запись `watch_record`, которая содержит следующие сведения: описание наблюдателя, время работы, нагрузку, результат условия наблюдателя.



Пользователь с привилегией `watcher_user` может просматривать наблюдатели и историю их работы.

Уведомления в деле

Теперь, когда вы знаете, из чего состоит наблюдатель, поговорим о том, как создавать и удалять наблюдатели, а также управлять ими. Для этих действий можно использовать следующее:

- пользовательский интерфейс наблюдателей в Kibana;
- REST API наблюдателей в X-Pack.

Пользовательский интерфейс наблюдателей использует REST API наблюдателей для управления ими. В этом разделе мы рассмотрим процесс создания и удаления наблюдателей, а также управления ими с помощью соответствующего интерфейса Kibana.

Создание нового уведомления

Для создания наблюдателя зайдите в Kibana (<http://localhost:5601>) как пользователь `elastic` и перейдите на страницу **Management** (Управление); щелкните на ссылке **Watcher** (Наблюдатель) в разделе **Elasticsearch** (рис. 8.28).

The screenshot shows the Kibana Management interface. On the left is a sidebar with icons for Discover, Visualize, Dashboard, Timeline, Machine Learning, Graph, Dev Tools, Monitoring, and Management. The Management icon is selected. The main area has a header 'Management' and 'Version: 6.0.0'. Below this are sections for Security, Users, Roles, Elasticsearch (with a 'Watcher' button highlighted by a red arrow), Kibana, Index Patterns, Saved Objects, and Reporting.

Рис. 8.28

При нажатии кнопки **Create New Watch** (Создать новый наблюдатель) вы увидите два варианта создания уведомлений (рис. 8.29):

- **Threshold Alert** (Пороговое уведомление);
- **Advanced Watch** (Продвинутое наблюдение).

The screenshot shows the 'Watches' page under 'Management / Elasticsearch / Watcher'. The sidebar is identical to Figure 8.28. The main area lists five existing watches. A 'Create new watch' button is visible. The first watch is a 'Threshold Alert' (highlighted by a red box) with the ID '0BwSPjgSQCmeRa...' and state 'Triggered' (a few seconds ago). The second watch is an 'Advanced Watch' with the ID '0BwSPjgSQCmeRa...' and state 'OK' (a few seconds ago). Both have a 'Delete' button next to them.

Рис. 8.29

Используя вариант **Threshold Alert** (Пороговое уведомление), можно создать уведомление, которое основано на определенном пороге для получения уведомлений, когда метрика выходит за пределы установленного порогового значения. На этой странице пользователи могут легко создавать уведомления порогового типа, не беспокоясь о необходимости

работать напрямую с сырьими запросами JSON. Данный пользовательский интерфейс позволяет создавать уведомления только для индексов на базе времени (то есть индексов с меткой времени).

С помощью варианта **Advanced Watch** (Продвинутое наблюдение) можно создавать наблюдатели, работая напрямую с `.json` для API наблюдателей.



Для работы с пользовательским интерфейсом наблюдателей, а именно для создания, редактирования, удаления и деактивизации наблюдателей, необходим пользователь с привилегиями `kibana_user` и `watcher_admin`.

Пороговое уведомление

Нажмите кнопку **Create New Watch** (Создать новый наблюдатель) и выберите вариант **Threshold Alert** (Пороговое уведомление). Вы попадете на страницу пороговых уведомлений.

Укажите название уведомления; выберите индекс, который будет запрашиваться, поле времени и частоту триггера в пользовательском интерфейсе порогового уведомления (рис. 8.30).

A screenshot of the Kibana interface showing the 'Management / Elasticsearch / Watcher / Watches / New Watch' section. The page title is 'New Watch'. Below it, a sub-section title is 'Create a new threshold alert'. A note says 'Send out an alert when specific conditions are met. This will run once every 30 seconds.' There are two input fields: 'Name' containing 'log_error_watch' and 'Select an index' containing 'log'. In the bottom right corner, there are two dropdown menus: 'Select a time field' set to 'timestamp' and 'Run this watch every' set to '30 seconds'.

Рис. 8.30

Далее необходимо указать условие, которое приведет к срабатыванию уведомления. Во время изменения выражений/условий визуализация обновляется автоматически и пороговое значение и данные демонстрируются в виде красной и синей линии соответственно (рис. 8.31).



Рис. 8.31

Теперь укажите действие, которое будет выполнено при совпадении условия. Для этого нажмите кнопку **Add new action** (Добавить новое действие). Вы можете создать одно из трех действий: уведомление по электронной почте, по Slack и запись событий в лог. Можно выбрать одно или несколько действий (рис. 8.32).

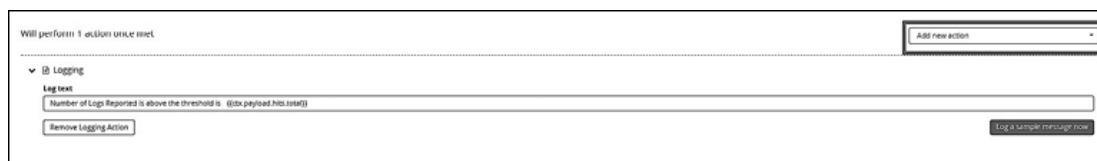


Рис. 8.32

В конце нажмите кнопку **Save** (Сохранить) для создания наблюдателя. Наблюдатель будет сохранен в индексе `watches`, как показано на рис. 8.33.

The screenshot shows a browser window with the URL `localhost:9200/.watches/_search`. The page title is "localhost:9200/.watches/_search". Below the title, there are two tabs: "Apps" (selected) and "New". The main content area displays a JSON document representing a watch configuration:

```

{
  "_index": ".watches",
  "_type": "doc",
  "_id": "237c7d43-c4a8-488f-9ed7-b168318ac1eb",
  "_score": 1,
  "_source": {
    "trigger": {
      "schedule": {
        "interval": "10s"
      }
    },
    "input": {
      "search": {
        "request": {
          "search_type": "query_then_fetch",
          "indices": [
            "logs"
          ],
          "types": [
            ""
          ],
          "body": {
            "size": 0,
            "query": {
              "bool": {
                "filter": [
                  {
                    "range": {
                      "timestamp": {
                        "gte": "{{ctx.trigger.scheduled_time}}|-1m",
                        "lte": "{{ctx.trigger.scheduled_time}}",
                        "format": "strict_date_optional_time||epoch_millis"
                      }
                    }
                  }
                ]
              }
            }
          }
        }
      }
    },
    "condition": {
      "script": {
        "source": "if (ctx.payload.hits.total > params.threshold) { return true; } return false;",
        "lang": "painless",
        "params": {
          "threshold": 20
        }
      }
    },
    "transform": {
      "script": {
        "source": "HashMap result = new HashMap(); result.result = ctx.payload.hits.total; return result;",
        "lang": "painless",
        "params": {
          "threshold": 20
        }
      }
    },
    "actions": {
      "logging_1": {
        "logging": {
          "level": "info",
          "text": "Number of Logs Reported is above the threshold is {{ctx.payload.result}}"
        }
      }
    },
    "metadata": {
      "name": "logs_errors_watch",
      "watcherui": {
        "trigger_interval_unit": "s",
        "agg_type": "count",
        "time_field": "timestamp",
        "trigger_interval_size": 10,
        "term_size": 5,
        "time_window_unit": "m",
        "threshold_comparator": ">",
        "term_field": null,
        "index": [
          "logs"
        ]
      }
    }
  }
}

```

Рис. 8.33

Продвинутое наблюдение

Нажмите кнопку **Create New Watch** (Создать нового наблюдателя) и выберите вариант **Advanced Watch** (Продвинутое наблюдение). Вы попадете на страницу настройки продвинутого наблюдения.

Укажите идентификатор и название наблюдателя и вставьте JSON для создания наблюдателя в области **Watch JSON** (Наблюдатель JSON), затем нажмите кнопку **Save** (Сохранить). Идентификатор наблюдателя присваивается в Elasticsearch при создании, тогда как для пользователя при работе с наблюдателями более удобным является название (рис. 8.34).



Рис. 8.34

Вкладка **Simulate** (Симуляция) предоставляет интерфейс для изменения частей наблюдателя с последующей проверкой их воспроизведения (рис. 8.35).

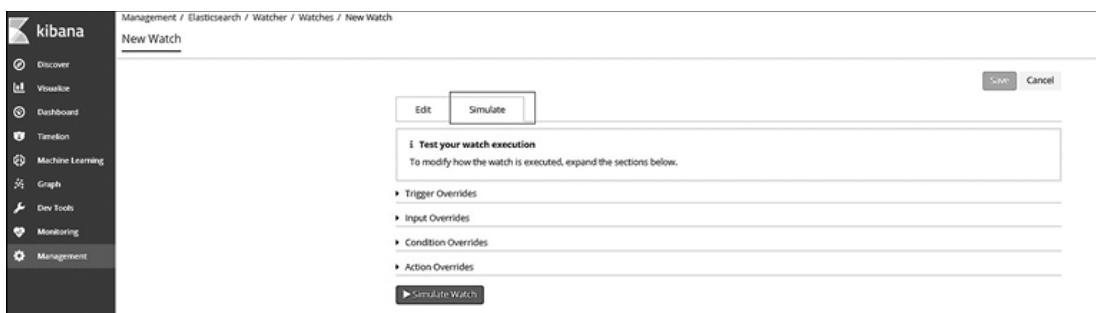


Рис. 8.35



Название наблюдателя будет сохранено в разделе метаданных тела наблюдателя. Можно использовать этот раздел при создании наблюдателя для хранения метаданных, тегов или иной информации.

После нажатия кнопки **Save** (Сохранить) наблюдатель будет сохранен в индексе `watches`, как показано на рис. 8.36.

```
{  
  "_index": ".watches",  
  "_type": "doc",  
  "_id": "logstash_error_watch",  
  "_score": 1,  
  "_source": {  
    "trigger": {  
      "schedule": {  
        "interval": "30s"  
      }  
    },  
    "input": {  
      "search": {  
        "request": {  
          "search_type": "query_then_fetch",  
          "indices": [  
            "logstash*"  
          ],  
          "types": [ ],  
          "body": {  
            "query": {  
              "match": {  
                "message": "error"  
              }  
            }  
          }  
        }  
      }  
    },  
    "condition": {  
      "compare": {  
        "ctx.payload.hits.total": {  
          "gt": 0  
        }  
      }  
    },  
    "actions": {  
      "log_error": {  
        "logging": {  
          "level": "info",  
          "text": "The number of errors in logs is {{ctx.payload.hits.total}}"  
        }  
      }  
    },  
    "metadata": {  
      "name": "Logstash Error Watch",  
      "type": "json"  
    },  
    "status": {...}  
  }  
},
```

Рис. 8.36

Как удалить/деактивизировать/редактировать наблюдатель

Для удаления наблюдателя перейдите в интерфейс **Management** (Управление) и выберите **Watches** (Наблюдатели) в разделе **Elasticsearch**. В списке наблюдателей выберите один или несколько объектов, которые следует удалить, и нажмите кнопку **Delete** (Удалить) (рис. 8.37).

ID	Name	State	Comment	Last Fired	Last Triggered
08wSPsjSQCmeRa...	X-Pack Monitoring...	✗ Firing		a few seconds ago	a few seconds ago
08wSPsjSQCmeRa...	X-Pack Monitoring...	✓ OK			a few seconds ago
08wSPsjSQCmeRa...	X-Pack Monitoring...	✓ OK			a few seconds ago
08wSPsjSQCmeRa...	X-Pack Monitoring...	✓ OK			a few seconds ago
08wSPsjSQCmeRa...	X-Pack Monitoring...	✓ OK			a few seconds ago
237c7d43-c4a8-49...	logs_errors_watch	✓ OK		a few seconds ago	Edit
<input type="checkbox"/> logstash_error_wa...	Logstash Error Wa...	✓ OK		a few seconds ago	Edit

1 watch selected

1-7 of 7 < >

Рис. 8.37

Для деактивизации наблюдателя (то есть временного отключения его работы) перейдите в интерфейс **Management** (Управление) и выберите **Watches** (Наблюдатели) в разделе **Elasticsearch**. В списке наблюдателей выберите необходимый объект. При этом вы увидите историю наблюдения. Нажмите кнопку **Deactivate** (Деактивизировать). На этой странице, как говорилось выше, также можно удалить наблюдатель.

Щелчок на времени выполнения (ссылке) в истории наблюдателя покажет детали конкретной записи `watch_record` (рис. 8.38).

Action	State	Comment
logging_1	✗ Firing	

Watch History

Trigger Time	State	Comment
December 11th 2017, 19:57:31.873	✗ Firing	
December 11th 2017, 19:57:12.367	✓ OK	
December 11th 2017, 19:57:01.865	✓ OK	
December 11th 2017, 19:56:51.360	✓ OK	
December 11th 2017, 19:56:40.850	✓ OK	
December 11th 2017, 19:56:30.348	✓ OK	
December 11th 2017, 19:56:19.847	✓ OK	
December 11th 2017, 19:56:09.823	✓ OK	
December 11th 2017, 19:55:59.321	✓ OK	
December 11th 2017, 19:55:48.818	✓ OK	
December 11th 2017, 19:55:38.314	✓ OK	

Рис. 8.38

Для редактирования наблюдателя перейдите на вкладку **Edit** (Редактирование) и внесите изменения. Для сохранения нажмите кнопку **Save** (Сохранить).

Резюме

Из этой главы вы узнали, как установить и сконфигурировать компоненты X-Pack в Elastic Stack и как настроить безопасность кластера путем создания пользователей и ролей. Вы также научились мониторить сервер Elasticsearch и получать уведомления, если данные изменяются или в них появляются ошибки.

В следующей главе мы соберем целое приложение, используя Elastic Stack для анализа данных и опираясь на все концепции, которые уже рассмотрели на данный момент.

9. Запуск Elastic Stack в работу

В процессе изучения Elastic Stack вы разобрали много тем и получили подробное представление обо всех его компонентах. Вы уже должны хорошо разбираться в ядре Elasticsearch и его возможностях поиска и аналитики, а также знаете, как эффективно применять Logstash и Kibana для создания мощной платформы аналитики больших данных. Помимо этого, вы видели, как легко настроить безопасность и мониторинг больших данных, а также получать уведомления, анализировать диаграммы и внедрять машинное обучение с помощью X-Pack.

Для включения Elastic Stack в рабочий процесс необходимо быть в курсе распространенных подводных камней и знать некоторые шаблоны и стратегии, которые помогут без проблем запустить решение. В этой главе мы дадим советы по запуску Elasticsearch, Logstash, Kibana и других компонентов в работу.

Мы начнем с Elasticsearch, а затем перейдем к другим составляющим. Есть несколько способов запуска Elasticsearch в эксплуатацию. Различные факторы могут влиять на то, какой именно способ выберете вы. Чтобы вам было проще запустить проект на базе Elastic Stack, мы рассмотрим следующие темы.

- Размещение Elastic Stack в управляемом облаке.
- Размещение Elastic Stack на своем хостинге.
- Создание и восстановление резервных копий.
- Настройка названий индексов.
- Настройка шаблонов индексов.

- Моделирование данных временного ряда.

Для начала разберемся, как запустить Elastic Stack, разместив его в одном из поставщиков управляемого облачного хостинга. Такое решение не потребует дополнительных действий для настройки готового к работе кластера.

Размещение Elastic Stack в управляемом облаке

Облачные поставщики позволяют многократно упростить процесс настройки готового к работе кластера. Как пользователю, вам не потребуется выполнять низкоуровневую настройку или выбирать аппаратную часть и управлять ею, операционной системой и многими другими параметрами конфигурации Elasticsearch и Kibana.

Существует несколько поставщиков облачных сервисов, которые предоставляют управляемые кластеры Elastic Stack: Elastic Cloud, QBox.io, Bonsai и многие другие. В этом разделе мы рассмотрим, как начать работу с *Elastic Cloud*. Elastic Cloud — это официальное предложение от компании Elastic.co — основного участника в развитии разработки Elasticsearch и других компонентов Elastic Stack. При работе с Elastic Cloud мы изучим следующие темы.

- Настройка и запуск Elastic Cloud.
- Использование Kibana.
- Изменение настроек.
- Восстановление.

Настройка и запуск Elastic Cloud

Зарегистрируйтесь в Elastic Cloud по ссылке <https://www.elastic.co/cloud/as-a-service/signup>; для этого понадобится адрес электронной почты и его подтверждение. Вам будет предложено установить исходный пароль.

После установки пароля вы можете зайти в консоль Elastic Cloud по ссылке <https://cloud.elastic.co>. Консоль Elastic Cloud предлагает простой пользовательский интерфейс для управления вашими кластерами. Поскольку вы зарегистрировались с учетной записью для пробного периода, вы получаете 4 Гбайт ОЗУ и 96 Гбайт дискового пространства на ограниченное время.

При запуске кластера мы можем выбрать *AWS (веб-сервисы Amazon)* или *GCE (движок вычислений Google)*. При входе вы можете создать кластер Elastic Cloud (рис. 9.1).

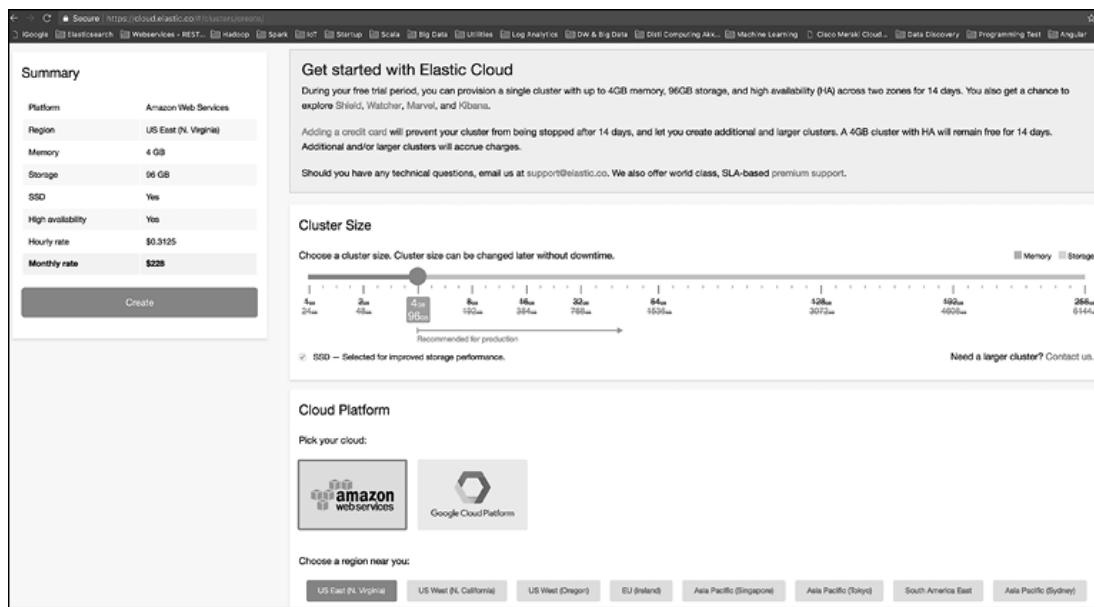


Рис. 9.1

Вы можете выбрать ОЗУ и дисковое пространство и решить, будете ли дублировать кластер в разных зонах доступности. Предоставляется также возможность настроить дополнительные

плагины, которые могут понадобиться вам в кластере Elasticsearch. Рекомендуется выбрать последнюю доступную версию 6.x. На момент написания этой книги версия 6.0.0 была новейшей доступной в Elastic Cloud.

Ведите название кластера и нажмите кнопку **Create** (Создать); ваш кластер будет создан и запущен с рабочей конфигурацией, в том числе безопасности. В базовую комплектацию также входит доступ Kibana. На данный момент вам будет предложено получить имя пользователя и пароль для входа в узлы Elasticsearch и Kibana. Запишите их. Вы также получите *идентификатор облака*, который пригодится вам при подключении облачного кластера к агентам beats и серверам Logstash.

При обзоре кластера вы должны увидеть следующую страницу (рис. 9.2).

The screenshot shows the Elastic Cloud web interface. At the top, there's a navigation bar with links like 'Cloud', 'Clusters', 'Help', and 'Sign out'. Below it, a sidebar on the left has buttons for 'New Cluster' and 'LearningECS'. Under 'LearningECS', there are tabs for 'Overview', 'Configuration', 'Logs', 'Security', and 'Snapshots'. The main content area is titled 'us-east-1'. It contains sections for 'Endpoints' (listing Elasticsearch and Kibana URLs), 'Dashboards' (with a small Kibana chart), and 'Node' (listing three nodes: 'elasticsearch-0000000000', 'instance-0000000000', and 'instance-0000000001', all in the 'running' status). A note at the bottom says: 'Your cluster is composed of one or more nodes. Nodes will be in different zones to ensure high availability. When your capacity grows big enough, there will be multiple nodes per zone.' Another note below that says: 'Memory pressure is the percentage used of the "old gen" memory pool. If this is high (>75%), expensive garbage collections will be more frequent. If memory pressure is high, the cluster should be upgraded to have more memory.'

Рис. 9.2

Как видите, кластер запущен и работает. Вместе с ним запущен компонент Kibana, доступ к которому возможен по

предоставленной URL-ссылке. Кластер Elasticsearch доступен по предоставленной безопасной ссылке HTTPS.

У этого кластера есть два узла: один в каждой зоне доступности AWS и один — разрешающий узел. Разрешающий узел необходим для выбора главного узла. Это особые узлы в Elastic Cloud, которые помогают в выборе нового главного узла в том случае, если отдельные узлы кластера становятся недоступными.

Теперь, когда все готово, приступим к работе!

Использование Kibana

Мы уже получили ссылку, по которой настроен доступ в Kibana, на странице обзора кластера в Elastic Cloud. Вы можете щелкнуть на ней для входа в интерфейс Kibana. В отличие от локального процесса Kibana, который мы создавали ранее, этот обладает защитой X-Pack. Для входа в систему вам необходимо указать имя и пароль, полученные ранее при создании кластера в Elastic Cloud, как описано в предыдущем разделе.

После входа в Elastic Cloud вы увидите интерфейс Kibana (рис. 9.3).



Рис. 9.3

На этой странице вы можете просматривать все индексы, анализировать данные вашего кластера Elasticsearch, а также выполнять их мониторинг.

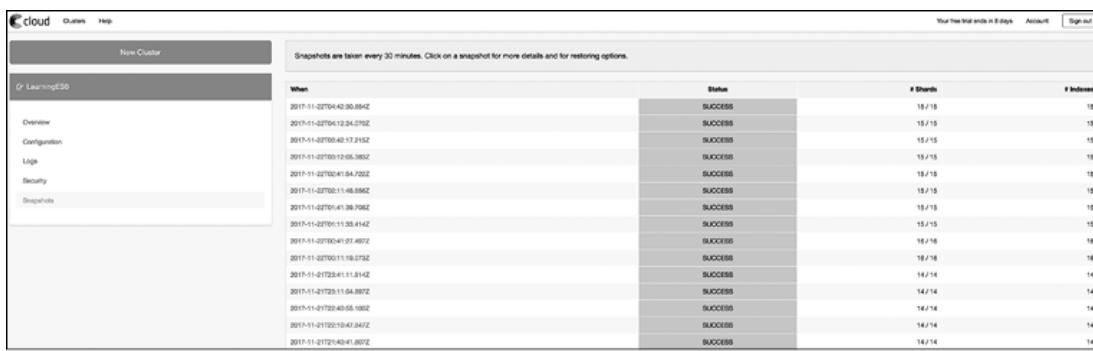
Изменение настроек

Вы можете указать свои настройки для узлов Elasticsearch на вкладке **Configuration** (Конфигурация) в Elastic Cloud. Нельзя напрямую редактировать файл `elasticsearch.yml` в Elastic Cloud. Тем не менее можно использовать раздел под названием **User Settings** (Пользовательские настройки), с помощью которого разрешается редактировать набор настроек конфигурации.

Полный список параметров конфигурации, которые могут быть изменены при использовании Elastic Cloud, вы найдете в документации по ссылке <https://www.elastic.co/guide/en/cloud/current/cluster-config.html#user-settings>.

Восстановление

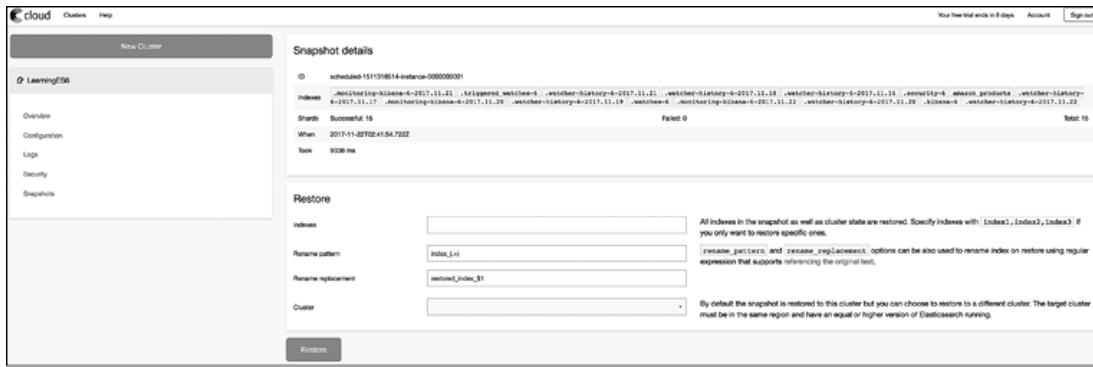
Elastic Cloud автоматически создает точку восстановления всех индексов вашего кластера с определенной периодичностью (каждые 30 мин). При необходимости можете восстановить свои данные. Процесс автоматизирован и не нуждается в дополнительных настройках или написании кода. Перейдите на вкладку **Snapshots** (Снепшоты), чтобы увидеть список доступных точек восстановления (снепшотов) (рис. 9.4).



When	Status	# Shards	# Indexes
2017-11-02T04:42:30.884Z	SUCCESS	18/18	18
2017-11-02T04:42:24.076Z	SUCCESS	19/19	19
2017-11-02T04:42:17.116Z	SUCCESS	19/19	19
2017-11-02T04:42:12.05386Z	SUCCESS	19/19	19
2017-11-02T04:41:54.722Z	SUCCESS	19/19	19
2017-11-02T02:11:46.086Z	SUCCESS	18/18	18
2017-11-02T01:41:39.706Z	SUCCESS	19/19	19
2017-11-02T01:11:38.414Z	SUCCESS	19/19	19
2017-11-02T00:41:27.497Z	SUCCESS	19/19	19
2017-11-02T00:11:01.079Z	SUCCESS	18/18	18
2017-11-01T23:41:11.814Z	SUCCESS	14/14	14
2017-11-01T23:11:04.897Z	SUCCESS	14/14	14
2017-11-01T22:40:54.106Z	SUCCESS	14/14	14
2017-11-01T22:10:41.847Z	SUCCESS	14/14	14
2017-11-01T21:40:41.897Z	SUCCESS	14/14	14

Рис. 9.4

Вы можете выбрать снепшот для восстановления и перейти на следующую страницу (рис. 9.5).



The screenshot shows the 'Snapshot details' section of the Elastic Cloud interface. It displays information about a specific snapshot, including its ID, creation time, shard count, and failure count. Below this, the 'Restore' section allows users to specify which indices to restore, using patterns like 'Index_1-h' and 'Index_1-h'. A note at the bottom states that the snapshot is restored to the current cluster.

Рис. 9.5

Снепшот восстановления содержит сохраненное состояние для всех индексов кластера. Вы можете выбрать несколько

индексов для восстановления, а также переименовать их в процессе. Восстановлению подлежат и некоторые кластеры.

Далее мы рассмотрим, как начать работу с Elastic Cloud, если вы планируете управлять компонентами Elastic Stack самостоятельно. Это называется *самостоятельным хостингом* в контексте того, что вы будете сами размещать систему и управлять ею.

Размещение Elastic Stack на своем хостинге

При самостоятельном хостинге Elastic Stack вы должны будете лично установить, настроить Elasticsearch и другие компоненты Elastic Stack и управлять ими. Это можно сделать двумя способами:

- используя внутренний хостинг;
- облачный хостинг.

Независимо от вашего выбора, то есть размещения Elastic Stack на внутреннем хостинге (ваш собственный датацентр) или в одном из облачных сервисов, таких как AWS, Azure или GCE, есть некоторые общие особенности, которые необходимо принимать во внимание. При самостоятельном хостинге вы столкнетесь со следующими задачами.

- Выбор аппаратной части.
- Выбор операционной системы.
- Настройка узлов Elasticsearch.
- Управление узлами Elasticsearch и их мониторинг.

- Особенности самостоятельного размещения в облаке.

Все вопросы, кроме последнего, имеют одинаковое решение, независимо от выбора облачного и собственного хостинга.

Выбор аппаратной части

Обычно Elasticsearch выполняет задачи обратного индекса, зависящие от памяти. Чем больше данных поместится в ОЗУ, тем выше будет производительность. Но это утверждение не работает для всех до единого случаев. Все зависит от типа ваших данных, операций над ними и объемов работы.

Не следует считать, что Elasticsearch выполняет все операции в памяти. Данные на диске также достаточно эффективно используются, особенно в случае агрегаций.



Все типы данных (кроме анализируемых строк) поддерживают особую структуру данных под названием `doc_values` – она организовывает данные на диске по принципу столбцов. Это полезно для операций сортировки и агрегации. Значение `doc_values` включено по умолчанию для всех типов данных, кроме анализируемых строк, следовательно, операции с ними работают по большей части с диска. Нет необходимости загружать эти поля в память для сортировки или агрегации.

Поскольку Elasticsearch может масштабироваться горизонтально, не так сложно сделать правильный выбор. Для начала лучше выбрать узлы с объемом ОЗУ от 16 до 32 Гбайт и около восьми ядер ЦП. Как вы увидите в следующих разделах,

размер кучи JVM в Elasticsearch не может превышать 32 Гбайт; следовательно, нет смысла применять конфигурации с более чем 64 Гбайт ОЗУ. Рекомендуется использовать твердотельные SSD-диски, если вы планируете выполнять сложные агрегации.

Важным моментом является запуск сравнительных тестов (бенчмарков) для проверки производительности на текущем оборудовании, а добавлять или совершенствовать узлы можно по необходимости.

Выбор операционной системы

Предпочтительной операционной системой для запуска Elasticsearch и других компонентов Elastic Stack является Linux. Ваш выбор ОС будет зависеть в основном от технологий, используемых в вашей компании. Если организация использует ПО от Microsoft, вы можете запустить Elastic Stack и на Windows.

Настройка узлов Elasticsearch

Elasticsearch, который является сердцем Elastic Stack, нуждается в предварительной настройке. Большинство настроек можно оставить со стандартными значениями, но есть показатели, которые необходимо подобрать на уровне ОС или JVM.

Размер кучи JVM

Установите одинаковое значение параметров `-Xms` и `-Xmx`. Большой размер кучи позволит Elasticsearch держать в памяти больше данных для быстрого доступа к ним. Но есть и обратная сторона: когда куча JVM будет близка к заполнению, сборщик мусора JVM начнет полную сборку мусора. В этот момент все

остальные процессы в Elasticsearch будут приостановлены. Чем больше размер кучи, тем длительнее будет эта пауза. Максимальный возможный размер кучи составляет около 32 Гбайт.

Не забывайте также, что следует распределить не более чем 50 % доступного ОЗУ для Elasticsearch JVM. Это делается для того, чтобы система имела в распоряжении достаточно памяти для работы кэша файловой системы Apache Lucene. Все данные, хранящиеся в узле Elasticsearch, рассматриваются как индексы Apache Lucene, которому необходима ОЗУ для быстрого доступа к файлам.

Таким образом, если вы планируете хранить огромные объемы данных в Elasticsearch, нет смысла иметь один узел с более чем 64 Гбайт ОЗУ (50 % от которой – 32 Гбайт – максимальный размер кучи). Рекомендуется добавлять больше узлов, если вы заинтересованы в масштабировании.

Отключение файла подкачки

При включенном файле подкачки ОС нередко имеет свойство выгружать память, использованную приложением, на диск, чтобы другие программы могли задействовать больше ОЗУ.

В узле Elasticsearch это может привести к тому, что ОС начнет выгружать память кучи Elasticsearch. Такой процесс записи содержимого ОЗУ на диск с последующей обратной записью с диска в ОЗУ может замедлить работу. По этой причине следует отключать файл подкачки в узле Elasticsearch.

Дескрипторы файлов

В операционных системах Linux и macOS ограничено количество дескрипторов открытых файлов, которые может хранить процесс. Зачастую необходимо увеличивать это

ограничение, так как значение по умолчанию слишком мало для работы Elasticsearch.

Пулы потоков и сборщик мусора

В Elasticsearch предусмотрено много видов операций, таких как индексирование, поиск, сортировка и агрегации. Кроме того, система использует пул потоков JVM для выполнения своих задач. Рекомендуется не менять настройки, связанные с пулами потоков в Elasticsearch, так как обычно это приводит скорее к негативным последствиям, нежели к улучшению производительности. Еще один компонент в Elasticsearch, настройки которого не стоит менять, — сборщик мусора.

Управление Elasticsearch и ее мониторинг

Когда вы используете самостоятельный хостинг Elasticsearch, все вопросы мониторинга и управления кластером ложатся на вас. Приходится мониторить статус процесса Elasticsearch, память и дисковое пространство в узле. Если узел по любой причине выходит из строя или становится недоступным, его необходимо запустить повторно.

Для регулярного резервного копирования необходимо делать снепшоты — точки восстановления индексов Elasticsearch. Мы еще обсудим функционал снепшотов для резервного копирования. Большую часть мониторинга можно выполнять средствами X-Pack и Kibana, но управление должно происходить вручную.

Запуск в контейнерах Docker

Docker — программное обеспечение для автоматизации развертывания и управления приложениями в средах с

поддержкой контейнеризации. Его преимущество состоит в том, что программное обеспечение упаковано внутри легковесного контейнера, который имеет минимум накладных расходов по сравнению с виртуальной машиной. В сочетании с этим и большим пулом общедоступных образов Docker является отличным способом запускать программное обеспечение в работу без особых настроек.

Официальные образы Elasticsearch в формате Docker доступны для скачивания в различных видах.

- Elasticsearch с базовой лицензией X-Pack.
- Elasticsearch с полной лицензией X-Pack и 30-дневным пробным периодом.
- Версия Elasticsearch с открытым исходным кодом без X-Pack.

Запуск Elasticsearch внутри контейнера Docker так же прост, как и установка Docker и выполнение команды `docker pull` с образом Elasticsearch на ваш выбор. Следующие простые команды запустят Elasticsearch 6.0.0 в одном узле, если в вашей системе установлен Docker.

```
docker pull
docker.elastic.co/elasticsearch/elasticsearch:6
.0.0
```

```
docker run -p 9200:9200 -p 9300:9300 -e
"discovery.type=single-node"
docker.elastic.co/elasticsearch/elasticsearch:6
.0.0
```

Подробнее узнать о запуске Elasticsearch в рабочей среде Docker вы можете в документации по следующей ссылке: <https://www.elastic.co/guide/en/elasticsearch/reference/6.0/docker.html>.

Особенности запуска в облаке

Для самостоятельного размещения Elastic Stack в облачном сервисе вы можете выбрать таких поставщиков, как AWS, Microsoft Azure, GCE и пр. Они предоставляют вычислительные и сетевые ресурсы, виртуальные частные облака и многое другое, что понадобится для контроля ваших серверов. В сравнении с запуском на собственных аппаратных ресурсах использование облачного поставщика имеет следующие преимущества.

- Нет необходимости инвестировать в аппаратное обеспечение.
- Можно повышать или понижать мощность серверов.
- Допускается при необходимости добавлять или удалять серверы.

При первом запуске обычно сложно понять, сколько именно ресурсов ЦП, ОЗУ необходимо. Облачный хостинг предоставляет вам гибкость, которая позволяет начать работу на одном типе конфигурации оборудования, а при необходимости — повысить или понизить вычислительную мощность или добавить/удалить узлы без дополнительных капиталовложений. В качестве примера мы рассмотрим EC2 и постараемся разобраться, что же стоит учитывать при выборе. В целом рекомендации будут одинаковыми независимо от

облачного поставщика. Далее приведен список вопросов, которые следует рассмотреть при выборе AWS EC2.

- Выбор типа реализации.
- Изменение портов. **Не оставляйте порты открытыми!**
- Запросы прокси.
- Связка HTTP с локальным адресом.
- Установка плагина исследования EC2.
- Установка плагина репозитория S3.
- Настройка периодического резервного копирования.

Рассмотрим эти вопросы по порядку.

Выбор типа реализации

EC2 предлагает разные типы реализаций для различных требований. Стандартным стартовым выбором для Elasticsearch считается реализация m3.2xlarge; она имеет восемь ядер ЦП, 30 Гбайт ОЗУ и два твердотельных диска на 80 Гбайт. Рекомендуется запускать бенчмарки и мониторить нагрузку на ресурсы ваших узлов. В зависимости от результатов вы можете сужать или расширять ресурсную базу.

Изменение портов по умолчанию. Не оставляйте порты открытыми!

Запуск любого сервиса в облаке сопряжен с определенными рисками. Важно учесть, что ни один из портов, используемых Elasticsearch, не должен быть доступен из Интернета. В EC2 вы

можете детально контролировать, какие порты доступны и с каких IP-адресов или подсетей. В целом нет необходимости открывать какие-либо порты для доступа извне, кроме порта 22 для удаленного доступа.

По умолчанию Elasticsearch использует порт 9200 для трафика HTTP и 9300 — для сообщения между узлами. Рекомендуется заменить эти порты по умолчанию другими, отредактировав файл `elasticsearch.yml` во всех ваших узлах.

Запросы прокси

Используйте обратные прокси, такие как nginx или Apache, для своих запросов к Elasticsearch/Kibana.

Связка HTTP с локальным адресом

Вам следует запускать узлы Elasticsearch в *VPC (Virtual Private Cloud)*. С недавних пор AWS создает все узлы в VPC. Узлы, которые не нуждаются в клиентском интерфейсе, принимают запросы от клиентов через HTTP. Это может быть определено параметром `http.host` в файле `elasticsearch.yml`. Вы можете больше узнать о связках хоста/порта HTTP в документации по ссылке <https://www.elastic.co/guide/en/elasticsearch/reference/current/modules-http.html>.

Установка плагина исследования EC2

Когда узлы Elasticsearch расположены в одной сети, они находят свои пиры через мультикаст. Это хорошо работает в обычной локальной сети. В случае с EC2 сеть является общей и автоматическое распознавание при связи узлов между собой не будет работать. Для этого необходимо установить плагин

исследования EC2 во всех узлах, что позволит анализировать новые узлы.

Для установки плагина исследования EC2 во все узлы выполните действия согласно инструкции по следующей ссылке:

<https://www.elastic.co/guide/en/elasticsearch/plugins/current/discovery-ec2.html>.

Установка плагина репозитория S3

При работе с Elasticsearch важно регулярно делать резервные копии, чтобы быстро восстановить данные в случае фатального сбоя или если требуется откатить систему до последнего исправного состояния. Более детально о сохранении и восстановлении резервных копий с помощью API снепшотов/восстановления Elasticsearch мы поговорим в следующем разделе. Чтобы резервные копии создавались регулярно и хранились в надежном централизованном месте, необходимо настроить механизм снепшотов. Если вы запускаете Elasticsearch на EC2, имеет смысл хранить снепшоты в сегменте AWS S3.



S3 расшифровывается как Simple Storage Service (простой сервис хранения). Он может масштабироваться и является надежным местом для хранения больших объемов данных. Предоставляет защиту для ваших данных и доступ с многих различных платформ. S3 соответствует самым строгим требованиям благодаря широким возможностям обеспечения безопасности. Нередко этот сервис является самым

предпочтительным решением для долгосрочного хранения данных, особенно когда системы, которые производят эти данные, размещены на AWS.

Плагин репозитория S3 может быть установлен с помощью следующей команды; его необходимо установить в каждый узел вашего кластера Elasticsearch:

```
sudo      bin/elasticsearch-plugin install  
repository-s3
```

Настройка периодического резервного копирования

Как только ваш репозиторий S3 настроен, важно убедиться, что периодически создаются резервные копии (снепшоты). Это значит, что нам необходимо запланированное задание, которое будет создавать точки восстановления с регулярными интервалами. Интервал может составлять 15, 30 мин, 1 ч и т.д. в зависимости от чувствительности данных. Мы рассмотрим процесс настройки резервного копирования и восстановления далее в этой главе.

На данном этапе мы разобрались, как запустить Elasticsearch в управляемом облаке или в условиях самостоятельного хостинга. Если вы выбрали последний вариант, придется настроить процесс резервного копирования и восстановления во избежание потерь данных. Информация из следующего раздела применима только в том случае, если вы используете собственный хостинг для кластера Elasticsearch.

Резервное копирование и восстановление

Наличие резервных копий очень важно, если понадобится восстановить данные в случае фатального сбоя. Настоятельно

рекомендуется сохранять все ваши данные с фиксированным интервалом времени и хранить достаточное количество резервных копий. Это обычная стратегия. Ваш кластер может быть расположен на внутреннем хостинге, в собственном dataцентре либо на облачном хостинге наподобие AWS с возможностью управлять кластером собственоручно.

В этом разделе мы рассмотрим следующие темы.

- Настройка репозитория для снепшотов.
- Создание снепшотов.
- Восстановление данных из выбранного снепшота.

Настройка репозитория для снепшотов

Первый шаг в создании регулярного резервного копирования данных — настройка репозитория для хранения снепшотов (точек восстановления). Их можно хранить в разных местах:

- в общей файловой системе;
- в облаке распределенных файловых систем (S3, Azure, GCS или HDFS).

Выбор способа хранения снепшотов зависит от того, где размещен ваш кластер Elasticsearch и какие параметры хранения доступны.

Сначала рассмотрим самый простой способ с использованием общей файловой системы.

Общая файловая система. Когда у вашего кластера есть общая файловая система, которая доступна со всех узлов кластера, вам нужно убедиться, что она доступна по общему

пути. Следует примонтировать выбранную общую папку во всех узлах и добавить путь к этой папке. Общий путь к монтированной файловой системе необходимо добавить в файл `elasticsearch.yml` в каждом узле следующим образом:

```
path.repo: ["/mount/es_backups"]
```



Если ваш кластер содержит единственный узел и вы не запустили полноценный распределенный кластер, нет необходимости настраивать общий диск. Параметр `path.repo` может быть указан и для локальной папки вашего узла. Запускать рабочий сервер в кластере из одного узла не рекомендуется.

Как только указанная строка добавлена в файл `config/elasticsearch.yml` во всех ваших узлах, их необходимо перезапустить.

Следующий шаг — регистрация названного репозитория под зарегистрированной папкой. Для этого выполняется команда `curl`, с помощью которой мы регистрируем названный репозиторий с названием `backups`:

```
curl -XPUT 'http://localhost:9200/_snapshot/backups' -H 'Content-Type: application/json' -d '{  
  "type": "fs",  
  "settings": {  
    "location": "/mount/es_backups/backups",  
    "compress": true  
}
```

```
}'
```

Вам потребуется заменить параметр `localhost` именем хоста или IP-адресом одного из узлов вашего кластера. Параметр `type` указан `fs` для общей файловой системы. Тело параметра `settings` зависит от значения параметра `type`.

Поскольку в данный момент мы ищем репозиторий общей файловой системы, тело параметра `settings` должно быть настроено особым образом. Параметр `location` указан в виде абсолютного пути, он должен вести к одной из папок, которые зарегистрированы с параметром `path.repo` в файле `elasticsearch.yml`. Если `location` не является абсолютным, Elasticsearch посчитает его относительным путем из параметра `path.repo`. Параметр `compress` сохраняет снепшоты в сжатом виде.

Облако распределенных файловых систем

Если вы запустили кластер Elasticsearch в сервисе AWS, Azure или Google Cloud, имеет смысл хранить точки восстановления (снепшоты) в альтернативном месте, предоставляемом облачной платформой. Такое хранилище будет более надежным и устойчивым к сбоям, чем общий диск.

Elasticsearch имеет официальные плагины, которые позволяют вам хранить снепшоты в S3. Все, что вам необходимо сделать, — установить плагин репозитория S3 во все узлы вашего кластера и настроить репозиторий по аналогии с тем, как мы настраивали репозиторий общей файловой системы:

```
curl -XPUT 'http://localhost:9200/_snapshot/backups' -H 'Content-Type:
```

```
application/json' -d '{  
    "type": "s3",  
    "settings": {  
        "bucket": "bucket_name",  
        "region": "us-west",  
        ...  
    }  
}'
```

В качестве `type` должен быть указан `s3`, а параметр `settings` должен иметь соответствующие значения для `s3`.

Создание снепшотов

Как только репозиторий настроен, мы можем создавать снепшоты с названиями на основе определенных репозиториев:

```
curl -XPUT  
'http://localhost:9200/_snapshot/backups/backup  
_201710101930?pretty' -H  
'Content-Type: application/json' -d'  
{  
    "indices": "bigginsight,logstash-*",  
    "ignore_unavailable": true,  
    "include_global_state": false  
}'
```

В этой команде мы указали, что хотим создать снепшот в репозитории `backups` с именем `backup_201710101900`. Имя снепшота может быть любым, но оптимально его сделать таким, чтобы оно в дальнейшем помогало в идентификации. Одна из типичных стратегий — настройка создания снепшота

каждые 30 минут с именем, содержащим префиксы вида `backup_yyyyMMddHHmm`. При неполадках вы сможете легко идентифицировать необходимый файл для восстановления.

При настройках по умолчанию снепшоты действуют по нарастанию. Лишние данные в них не хранятся.

Не исключено, что при периодической записи снепшотов вам понадобится просмотреть список всех имеющихся точек восстановления. Для этого выполните следующую команду:

```
curl -XGET  
'http://localhost:9200/_snapshot/backups/_all?  
pretty'
```

Восстановление из определенного снепшота

При необходимости вы можете выбрать определенную точку восстановления с помощью следующей команды:

```
curl -XPOST  
'http://localhost:9200/_snapshot/backups/backup  
_201710101930/_restore'
```

Таким образом мы восстановим систему из снепшота `backup_201710101930` из репозитория резервного копирования.

Настроив периодичность создания снепшотов, мы обеспечили безопасность на случай возникновения любых неполадок. У нас есть кластер, который можно восстановить в любой критической ситуации. Помните, что вывод снепшотов должен находиться в надежном хранилище. Как минимум не стоит хранить их в том же кластере Elasticsearch; необходимо выбрать другое хранилище, лучше всего надежную файловую систему с высоким уровнем доступности, такую как S3, HDFS и пр.

На данный момент в этой главе мы запустили кластер высокого уровня надежности с регулярным резервным копированием. В следующих разделах мы рассмотрим, как действовать в некоторых распространенных ситуациях моделирования данных. Мы разберем несколько базовых стратегий для настройки псевдонимов индексов, шаблонов индексов, моделирования данных временного ряда и пр.

Присвоение псевдонимов индексам

Псевдонимы индексов позволяют создавать псевдоним для одного или нескольких индексов или шаблонов названий индексов. Чтобы узнать принцип работы псевдонимов, мы рассмотрим следующие темы.

- Что такое псевдонимы индексов.
- Чем могут помочь псевдонимы индексов.

Что такое псевдонимы индексов

Псевдоним индекса предоставляет дополнительное название для индекса; его можно указать следующим образом:

```
POST /_aliases
{
  "actions" : [
    { "add" : { "index" : "index1", "alias" :
      "current_index" } }
  ]
}
```

В данном случае `index1` можно найти по псевдониму `current_index`. Аналогичным образом допускается удалить псевдоним индекса, используя действие `_aliases` в REST API:

```
POST /_aliases
{
  "actions" : [
    { "remove" : { "index" : "index1", "alias"
: "current_index" } }
  ]
}
```

Выше вы видите, как удалить псевдоним `current_index`. В одном вызове в API `_aliases` можно скомбинировать два действия. При комбинировании операции выполняются автоматически. Например, следующий вызов будет абсолютно прозрачным для клиента:

```
POST /_aliases
{
  "actions" : [
    { "remove" : { "index" : "index1", "alias"
: "current_index" } },
    { "add" : { "index" : "index2", "alias" :
"current_index" } }
  ]
}
```

До вызова псевдоним `current_index` ссылался на индекс `index1`, а после вызова будет ссылаться на `index2`.

Чем могут помочь псевдонимы индексов

После запуска сервера в работу возможна ситуация, когда необходимо переиндексировать данные из одного индекса в другой. Возможно, у нас есть программы, разработанные на Java, Python, .NET или в определенных программных средах, а в них — ссылки на эти индексы. Если индексы изменятся с `index1` на `index2`, потребуется вносить соответствующие изменения во всех клиентских приложениях.

В таких ситуациях пригодятся псевдонимы. Благодаря им достигается дополнительная гибкость, которую рекомендуется использовать в работе. Ключевым фактором является назначение вашему рабочему индексу псевдонима, который будет использован в настройках ваших приложений вместо фактического названия индекса.

Если требуется изменить текущий рабочий индекс, мы просто обновим настройки псевдонима так, чтобы он ссылался на новый индекс вместо старого. Используя эту функцию, мы можем избежать потерь времени при миграции данных или переиндексировании. Псевдонимы работают по известному в компьютерной науке принципу: дополнительный уровень косвенности может решить большинство проблем (более детально — по ссылке <https://en.wikipedia.org/wiki/Indirection>).

Здесь описаны не все функции, предоставляемые псевдонимами; среди них есть такие, как возможность использования шаблонов индекса, маршрутизация, возможность указывать фильтры и многое другое. Мы рассмотрим псевдонимы индексов и создание временных индексов далее в этой главе.

Настройка шаблонов индексов

Важным шагом в настройке вашего индекса является определение разметки типов, количества шардов, реплик и

другие настройки. В зависимости от сложности типов в вашем индексе этот шаг может потребовать существенного количества настроек.

Шаблоны позволяют вам создавать индексы по уже выбранному образцу, избавляя от необходимости создавать каждый индекс вручную. С помощью шаблонов индексов вы можете указать настройки и разметку для создаваемого индекса. Для понимания процесса выполним следующие шаги.

1. Определение шаблона индекса.

2. Создание индексов на ходу.

Предположим, что нам нужно хранить данные датчиков различных устройств и требуется каждый день создавать новый индекс. В начале каждого дня должен создаваться индекс — каждый раз, когда индексируется первая запись показаний датчика. Далее мы рассмотрим подробнее этот пример, а также причину использования временных индексов.

Определение шаблона индекса

Начнем с определения шаблона индекса:

```
PUT  
_template/readings_template  
{  
    "index_patterns":  
        ["readings*"],  
    "settings":  
        {"  
            "number_of_shards": 1  
        },  
    "mappings":  
        {  
            "readings":  
                {  
                    "properties":  
                        {"  
                            "date":  
                                {  
                                    "type": "date"  
                                }  
                            ,  
                            "id":  
                                {  
                                    "type": "string"  
                                }  
                            ,  
                            "value":  
                                {  
                                    "type": "float"  
                                }  
                        }  
                }  
        }  
}
```

```

        "mappings": 4
{
    "reading": {
        "properties": {
            "sensorId": {
                "type": "keyword"
            },
            "timestamp": {
                "type": "date"
            },
            "reading": {
                "type": "double"
            }
        }
    }
}
}

```

В этом вызове `_template` мы определили следующие настройки.

1. Шаблон с названием `readings_template`.
2. Шаблоны названий индекса, которые соответствуют этому шаблону. Мы создали `readings*` как единственный шаблон индекса. При любой попытке записи в индекс, которого не существует, но который соответствует такому шаблону, будет использован данный шаблон.
3. Настройки, которые будут применяться к созданному индексу из этого шаблона.

4. Разметка, которая будет применяться к созданному индексу из шаблона.

Попробуем занести данные в этот новый индекс.

Создание индекса на ходу

Если любой клиент пытается индексировать данные для определенного датчика устройства, должно использоваться название индекса с добавлением к нему значения текущего дня в формате `yyyy-mm-dd` (указывается после параметра `readings`). Вызов данных индекса для `2017-01-01` будет выглядеть следующим образом:

```
POST /readings-2017-01-01/reading
{
  "sensorId": "a11111",
  "timestamp": 1483228800000,
  "reading": 1.02
}
```

Когда вставляется первая запись для даты `2017-01-01`, клиент должен использовать название индекса `readings-2017-01-01`. Поскольку этот индекс еще не существует, а мы создали соответствующий шаблон, Elasticsearch создает новый индекс с помощью указанного шаблона. В результате разметка и настройки, которые мы определили, применяются к этому индексу.

Таким образом создаются индексы на базе шаблонов. В следующем разделе мы рассмотрим, почему эти типы временных индексов полезны и как их использовать в работе с данными временного ряда.

Моделирование данных временного ряда

Нередко приходится хранить в Elasticsearch данные временного ряда. Обычно создается один индекс для всех документов. Ситуация, когда один огромный индекс содержит все документы, является типичным примером, который имеет свои ограничения, особенно учитывая следующие причины.

- Масштабирование индекса с непредсказуемым объемом через некоторое время.
- Изменение разметки со временем.
- Автоматическое удаление более старых документов.

Рассмотрим проявление каждой проблемы при выборе одного монолитного индекса.

Масштабирование индекса с непредсказуемым объемом через некоторое время

Одно из самых сложных решений при создании кластера Elasticsearch и его индексов — выбор количества основных шардов и количества его реплик.

Разберемся, когда количество шардов становится важным.

Единица параллельности в Elasticsearch

Во время создания индекса требуется определить количество шардов. После создания индекса это значение изменить невозможно. Единого золотого правила для определения верного количества шардов на момент создания индекса не существует. Фактически количество шардов влияет на уровень

параллельности в индексе. Разберемся в этом на примере того, как может быть выполнен поисковый запрос.

Когда клиент отправляет запрос поиска или агрегации, сначала его получает один из узлов кластера. Этот узел работает как координатор запроса. Координирующий узел отправляет запросы на все шарды кластера и ждет ответа от каждого из них. Как только ответы получены, координирующий узел собирает их в единый файл и отсылает изначальному клиенту.

Это означает, что чем больше у вас шардов, тем меньше работы придется выполнять каждому из них и уровень параллельности может быть увеличен.

Но можно ли выбрать произвольно большое количество шардов? Рассмотрим это в следующих подразделах.

Как количество шардов влияет на оценку релевантности

Выбор большого количества малых шардов не всегда является хорошим решением, поскольку может повлиять на релевантность в результатах поиска. В контексте поисковых запросов оценка релевантности рассчитывается внутри контекста шарда. Относительный процент встречаемости документов считается в пределах контекста каждого шарда, а не по всем шардам. Поэтому количество шардов может влиять на общую оценку в рамках запроса. В частности, не стоит использовать слишком много шардов для решения проблемы масштабирования в будущем.

Как количество шардов влияет на точность агрегаций

По аналогии с поисковым запросом запрос агрегации также управляет с помощью узла координации. Предположим, клиент запросил агрегацию терминов для поля, которое может

содержать большое количество уникальных значений. По умолчанию агрегация терминов возвращает клиенту десять наиболее подходящих (топ) терминов.

Для координации выполнения терм-запроса узел координации не запрашивает все сегменты из всех шардов. Шарды опрашиваются для получения лучших n сегментов. По умолчанию значение n равно параметру `size` агрегации терминов, то есть количеству лучших сегментов, которые запросил клиент. Таким образом, если клиент запросил топ-10 терминов, координирующий узел вернет топ-10 сегментов из каждого шарда.

Поскольку данные могут быть распределены по шардам особым образом, некоторые из шардов могут не иметь определенных сегментов, даже несмотря на то, что эти сегменты могут входить в топ в некоторых шардах. Если определенный сегмент входит в топ- n сегментов в одном из шардов и не входит в топ- n в другом, он не будет учитываться в общем счете агрегации в координирующем узле. Большое количество шардов используется для масштабирования в будущем, но не увеличивает точность агрегаций.

Теперь вы понимаете, почему важно правильно выбрать количество шардов и почему это сложное решение. Далее мы рассмотрим, как изменение разметки индексов усложняется со временем.

Изменение разметки со временем

Как только индекс создан и в нем начинают сохраняться документы, требования к нему могут меняться.

Если схема меняется, с данными может происходить следующее.

- Добавление новых полей.
- Удаление существующих полей.

Добавление новых полей

Когда первый документ с новым полем индексируется, разметка нового поля создается автоматически, если она еще не существует. Для создания разметки Elasticsearch определяет тип данных для поля, основываясь на значении этого поля в первом документе. Разметка для одного конкретного типа документов со временем может увеличиваться.

Как только документ с новым полем проиндексирован, разметка для этого нового поля создается и сохраняется.

Удаление существующих полей

С течением времени требования к проекту могут меняться. Некоторые поля могут перестать иметь значение и не использоваться. В случае с индексами Elasticsearch не используемые более поля не удаляются автоматически; разметка продолжает находиться в индексе для всех полей, которые когда-либо были проиндексированы. Каждое дополнительное поле в Elasticsearch потребляет ресурсы; это особенно актуально, если у вас сотни или тысячи файлов. Если у вас очень большое количество неиспользуемых полей, это может привести к увеличению нагрузки на кластер.

Автоматическое удаление более старых документов

Ни в одном кластере нет бесконечного ресурса для вечного хранения данных. Если объемы ваших данных растут, вы можете захотеть хранить в Elasticsearch только необходимые

данные. Обычно в Elasticsearch предпочтительно хранить данные за последние несколько недель, месяцев или лет, в зависимости от случая.

До версии Elasticsearch 2.x это достигалось благодаря указанию в отдельных документах времени жизни (*time to live, TTL*). Предоставлялась возможность настроить время нахождения каждого документа в индексе. Но функция TTL была удалена из версии 2.x и далее по причине того, что накладно настраивать TTL для каждого документа.

Мы обсудили проблемы, с которыми можно столкнуться при работе с данными временного ряда. Теперь рассмотрим, как использовать *временные индексы* для разрешения этих проблем. Временные индексы также называются *индексом «в период времени»*.

Как временный индекс решает эти проблемы

Вместо того чтобы работать с одним монолитным индексом, теперь мы можем создать один индекс на период времени. Периодом может быть один день, неделя, месяц или произвольный отрезок времени. Например, в нашем случае в разделе **Index Template** (Шаблоны индексов) мы выбрали индекс «в день». Названия индексов будут выглядеть соответственно — у нас были индексы с названиями `readings-2017-01-01`, `readings-2017-01-02` и т.д. Если бы мы выбрали индекс «в месяц», названия выглядели бы следующим образом: `readings-2017-01`, `readings-2017-02`, `readings-2017-03` и т.д.

Посмотрим, как эта схема решает проблемы, перечисленные ранее.

Масштабирование с временным индексом

Поскольку мы более не используем монолитный индекс, который хранит все-все данные, упрощается масштабирование вверх/вниз. Количество шардов не выбирается заранее и навсегда. Начните с предполагаемого начального количества шардов для выбранного периода времени. Это количество можно задать в шаблоне индекса.

С каждым периодом времени вы имеете шанс подкорректировать шаблон индекса: увеличить или уменьшить количество шардов при создания следующего индекса.

Изменение разметки со временем

Изменять разметку становится проще, поскольку мы можем просто обновить шаблон индекса, который используется для создания новых индексов. Когда шаблон индекса обновлен, новый индекс создается для нового периода времени и, следовательно, применяется новая разметка.

С каждым новым периодом времени есть возможность внести изменения.

Автоматическое удаление старых документов

При использовании временных индексов удаление старых документов становится проще. Мы можем просто отсечь старые индексы вместо того, чтобы удалять отдельные документы. Если бы мы использовали месячные индексы и хотели сохранять данные на протяжении шести месяцев, то могли бы удалить все индексы старше шести месяцев. Поиск и удаление старых индексов можно настроить по расписанию.

Как вы уже видели в этом разделе, настройка индекса «в период времени» имеет очевидные преимущества при работе с данными временного ряда.

Резюме

В этой главе мы обсудили важные техники, которые пригодятся при запуске в работу вашего следующего приложения на базе Elastic Stack. Мы рассмотрели различные варианты размещения, включая собственный и облачный хостинг. Мы разобрались в использовании управляемого облачного сервиса, такого как Elastic Cloud, а также в размещении Elastic Stack на самостоятельном хостинге. Разобрали некоторые распространенные проблемы и их решения, имеющие отношение к обоим типам размещения вашего кластера.

Дополнительно мы рассмотрели различные методы, которые могут быть полезными при запуске Elastic Stack в эксплуатацию. Среди них использование псевдонимов индексов, шаблонов индексов, моделирование данных временного ряда. Разумеется, это не всеобъемлющее руководство, которое могло бы охватить все нюансы внедрения Elastic Stack. Но в книге приведено достаточно информации, чтобы вы могли успешно запустить проект на базе Elastic Stack в работу.

В следующей главе на базе полученных знаний мы построим приложение для аналитики данных датчиков.

10. Создание приложения для анализа данных с датчиков

Из предыдущей главы вы узнали, как можно запустить в работу приложение на базе Elastic Stack. Теперь попробуем применить эти концепции на практике, при работе с реальным приложением. В этой главе вы собственноручно создадите такое приложение на базе Elastic Stack, которое способно обрабатывать большие объемы данных. И примените в процессе все знания, полученные ранее.

В этой главе мы рассмотрим следующие темы.

- Введение в приложение.
- Моделирование данных в Elasticsearch.
- Настройка базы метаданных.
- Создание контейнера данных Logstash.
- Отправка данных в Logstash через HTTP.
- Визуализация данных в Kibana.

Введение в приложение

IoT (Internet of Things – «Интернет вещей») стал причиной появления широкого спектра приложений. Он может быть определен следующим образом:

Интернет вещей (ИВ) является коллективной сетью соединенных между собой смарт-устройств, которые могут общаться между собой, передавая данные и обмениваясь информацией через Интернет.

Устройства ИВ подключены к Интернету, снабжены различными типами датчиков, которые собирают данные и передают их по

Сети. Эти данные можно хранить, анализировать и изменять в режиме реального времени. Количество подключенных устройств будет стремительно расти; согласно данным «Википедии», к 2020 году ожидается 30 миллиардов подключенных устройств. Поскольку каждое устройство может получать текущее значение метрики и транслировать его через Интернет, результатом будут большие объемы данных.

В последнее время появилось большое количество типов датчиков, которые служат для измерения температуры воздуха, влажности, света, движения; они могут использоваться в различных приложениях. Каждый датчик может быть запрограммирован для считывания текущих показателей и отправки их по Сети.

Рассмотрим рис. 10.1, на котором показано, как подключенные устройства и датчики отправляют данные в Elastic Stack.

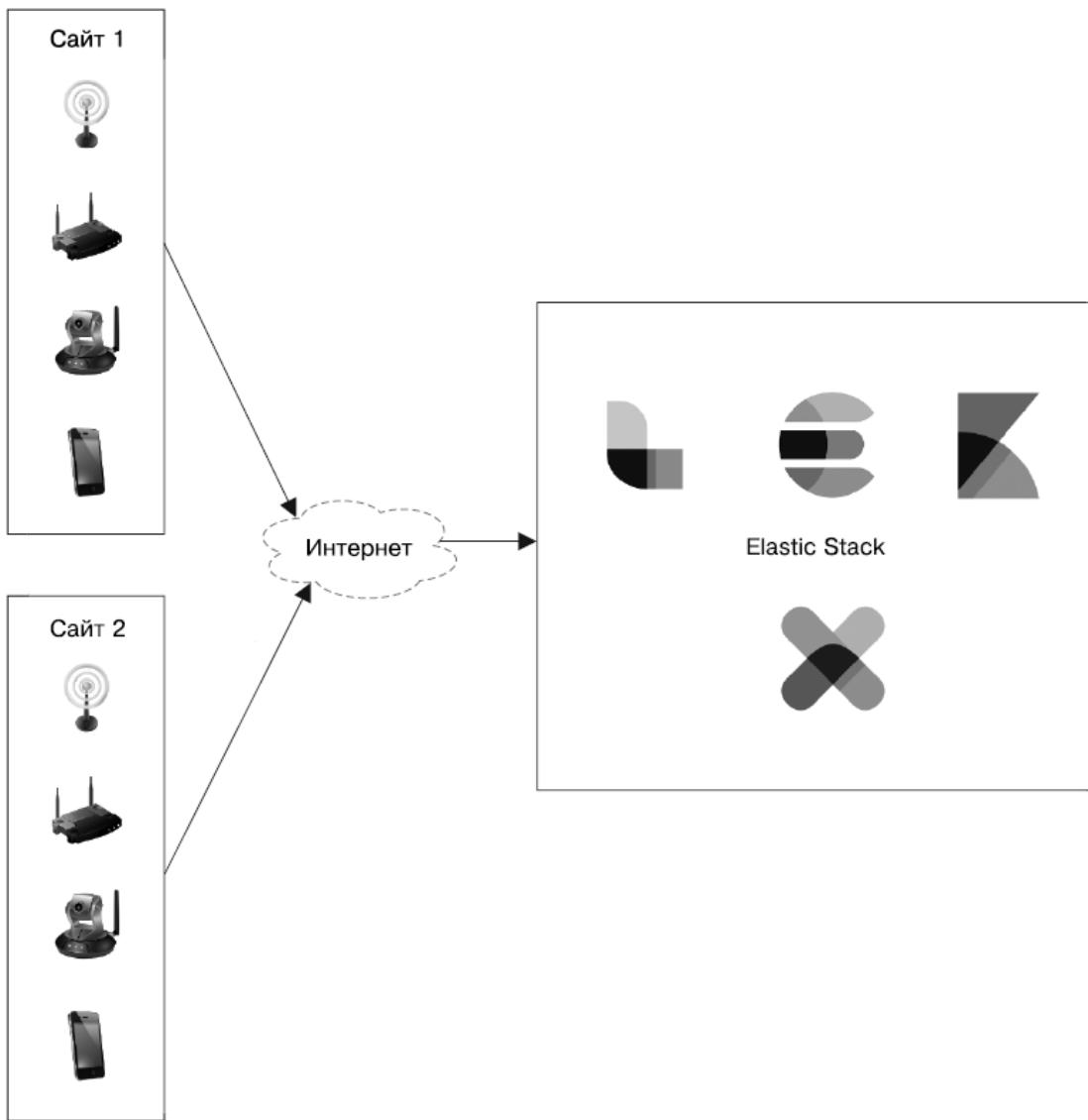


Рис. 10.1

Вы можете увидеть, как выглядит высокоуровневая архитектура системы, которую мы будем обсуждать в этой главе. В левой части рисунка представлены различные типы устройств, оснащенных датчиками. Эти устройства могут получать разные метрики и отправлять их через Интернет для долгосрочного хранения и анализа. Справа вы видите компоненты серверной части. В основном они принадлежат Elastic Stack.

В этой главе мы рассмотрим приложение для хранения и анализа данных, поступающих с двух типов датчиков: температуры

и влажности.

Датчики размещены в различных местах или на площадках, каждое место имеет подключение к Интернету (см. рис. 10.1). Наш пример демонстрирует работу с двумя типами датчиков, но приложение может быть расширено для поддержки датчиков любых типов.

В этом разделе мы рассмотрим следующие темы.

- Создаваемые датчиками данные.
- Метаданные с датчиков.
- Окончательно сохраняемые данные.

Создаваемые датчиками данные

Как выглядят данные, которые создаются датчиками? Датчик отправляет данные в формате JSON через Интернет, и каждая запись выглядит следующим образом:

```
{  
    "sensor_id": 1,  
    "time": 1511935948000,  
    "value": 21.89  
}
```

На примере мы видим следующее.

- Поле `sensor_id` является уникальным идентификатором датчика, который выполнил запись.
- Поле `time` является временем считывания показаний в миллисекундах, начиная с времени эпохи, то есть 00:00:00 1 января 1970 года.

- Поле `value` представляет собой фактическое метрическое значение, считанное датчиком.

Полезные данные JSON такого типа создаются каждую минуту всеми датчиками в системе. Поскольку все из них зарегистрированы в системе, со стороны сервера система имеет соответствующие для каждого датчика метаданные. Взглянем на метаданные, которые относятся к датчикам и доступны для нас из базы данных на стороне сервера.

Метаданные с датчиков

Метаданные, относящиеся ко всем датчикам на всех площадках, доступны для нас в реляционной базе данных. В нашем примере мы сохранили их в MySQL. Метаданные такого типа можно хранить в любой другой реляционной базе. Вы также можете добавить их в индекс Elasticsearch.

В большинстве случаев метаданные с датчиков содержат следующую информацию.

- **Тип датчика.** Что именно считывает датчик? Это может быть датчик температуры, влажности и т.д.
- **Метаданные о местонахождении.** Где физически находится датчик с выбранным идентификатором? С каким клиентом он связан?

Информация хранится в следующих трех таблицах MySQL.

- `sensor_type` — указывает различные типы датчиков и их `sensor_type_id`.

sensor_type_id	sensor_type

1	Температура
2	Влажность

- **location** — указано местонахождение с шириной/долготой и физическим адресом здания.

location_id	customer	department	building_name	room	floor	location_on_floor	latitude	longitude
1	Abc Labs	R & D	222 Broadway	101	1	C-101	40.710936	-74.008500

- **sensors** — связывает **sensor_id** с типом датчика и его местонахождением.

sensor_id	sensor_type_id	location_id
1	1	1
2	2	1

Имея представление о структуре базы данных, можно искать все метаданные, связанные с выбранным **sensor_id**, используя следующий запрос SQL:

```

select
    st.sensor_type as sensorType,
    l.customer as customer,
    l.department as department,
    l.building_name as buildingName,
    l.room as room,
    l.floor as floor,
    l.location_on_floor as locationOnFloor,
    l.latitude,
    l.longitude
from
    sensors s

```

```

        inner join
            sensor_type st ON s.sensor_type_id =
st.sensor_type_id
        inner join
            location l ON s.location_id = l.location_id
    where
        s.sensor_id = 1;

```

Результат предыдущего запроса будет выглядеть так.

sensorType	customer	department	buildingName	room	floor	locationOnFloor	latitude	longitude
Temperature	Abc Labs	R & D	222 Broadway	101	Floor1	C-101	40.710936	-74.0085

На данном этапе мы разобрались с форматом входящих данных датчиков с клиентской стороны. Кроме того, мы установили механизм получения связанных метаданных для выбранного датчика.

Далее мы рассмотрим, как должна выглядеть окончательная дополненная запись.

Окончательно сохраняемые данные

Комбинируя данные, поступающие с клиентской стороны и содержащие метрические значения с датчиков, мы можем создать дополненную запись из следующих полей.

1. sensorId.
2. sensorType.
3. customer.
4. department.
5. buildingName.

6. room.
7. floor.
8. locationOnFloor.
9. latitude.
10. longitude.
11. time.
12. reading.

Поля под номерами 1, 11 и 12 содержатся в данных, отправленных датчиком в наше приложение. Остальные поля созданы для дополнения, для этого используется запрос SQL, который мы видели в предыдущем разделе: `sensorId`. Таким образом мы можем создавать денормализованную запись для каждого датчика каждую минуту.

Итак, мы разобрались, в чем суть приложения и что собой представляют данные. Приступив к разработке приложения, мы начнем с самого центра. В случае с Elastic Stack в центре стека находится Elasticsearch, потому начнем разработку нашего решения с формирования модели данных. Этим мы и займемся в следующем разделе.

Моделирование данных в Elasticsearch

Для моделирования данных в Elasticsearch воспользуемся структурой финальной записи из предыдущего раздела. Учитывая, что у нас данные временного ряда, мы можем применить некоторые техники из главы 9.

- Определение шаблона индекса.

- Понимание разметки.

Взглянем на шаблон индекса, который будем определять.

Определение шаблона индекса. Поскольку мы будем хранить неизменяемые данные временного ряда, нет необходимости создавать один большой монолитный индекс. Мы будем использовать техники из раздела «Моделирование данных временного ряда» предыдущей главы.

Исходный код приложения, рассматриваемого в этой главе, находится в репозитории GitHub по следующей ссылке: <https://github.com/pranav-shukla/learningelasticsearch/tree/master/chapter-10>. На протяжении этой главы мы выполним шаги, перечисленные в файле README.md, который размещен по этому пути.

Создайте шаблон индекса, который упоминается в шаге 1 файла README.md, или запустите следующий скрипт в консоли Kibana (вкладка **Dev Tools** (Инструменты разработчика)):

```
POST _template/sensor_data_template
{
  "index_patterns": ["sensor_data*"],
  "settings": {
    "number_of_replicas": "1",
    "number_of_shards": "5"
  },
  "mappings": {
    "doc": {
      "properties": {
        "sensorId": {
          "type": "integer"
        },
        "sensorType": {
          "type": "keyword",
        }
      }
    }
  }
}
```

```
"fields": {
    "analyzed": {
        "type": "text"
    }
},
"customer": {
    "type": "keyword",
    "fields": {
        "analyzed": {
            "type": "text"
        }
    }
},
"department": {
    "type": "keyword",
    "fields": {
        "analyzed": {
            "type": "text"
        }
    }
},
"buildingName": {
    "type": "keyword",
    "fields": {
        "analyzed": {
            "type": "text"
        }
    }
},
"room": {
    "type": "keyword",
```

```
"fields": {
    "analyzed": {
        "type": "text"
    }
},
"floor": {
    "type": "keyword",
    "fields": {
        "analyzed": {
            "type": "text"
        }
    }
},
"locationOnFloor": {
    "type": "keyword",
    "fields": {
        "analyzed": {
            "type": "text"
        }
    }
},
"location": {
    "type": "geo_point"
},
"time": {
    "type": "date"
},
"reading": {
    "type": "double"
}
}
```

```
    }  
}  
}
```

Каждый раз, когда клиент пытается проиндексировать первую запись в этом индексе, шаблон индекса создаст новый индекс с названием по типу `sensor_data-YYYY.MM.dd`. Далее в разделе «Создание контейнера данных Logstash» мы рассмотрим, как можно сделать то же самое из Logstash.

Разбираемся с разметкой. Разметка, определенная в шаблоне индекса, содержит все поля, которые будут присутствовать в денормализованной записи после поиска. Обратите внимание на следующие особенности разметки шаблона индекса.

- Все поля, содержащие данные текстового типа, сохранены с типом `keyword`; дополнительно они сохраняются как `text` в анализируемом поле. Например, взгляните на поле `customer`.
- Поля широты и долготы, которые присутствовали в дополненных данных, теперь размечены как тип поля `geo_point`, название поля — `location`.

На данном этапе мы определили шаблон индекса, который будет запускать создание индексов с той разметкой, которую мы указали в шаблоне.

Настройка базы метаданных

Нам понадобится база данных, которая хранит метаданные с датчиков. В ней будут находиться таблицы, которые мы рассматривали в предыдущем разделе.

Мы храним данные в реляционной базе данных MySQL, но можно использовать любую другую реляционную базу данных. Исходя из нашего выбора, для подключения к базе данных мы

возьмем драйвер MySQL JDBC. Убедитесь, что в вашей системе присутствуют следующие компоненты.

1. База данных MySQL, версия от сообщества 5.5, 5.6 или 5.7. Вы можете использовать существующую базу данных, если уже настроили ее в системе.
2. Установите скачанную базу данных и зайдите в систему с пользователем **root**. Выполните сценарий, который находится по адресу https://github.com/pranav-shukla/learningelasticstack/tree/master/chapter-10/files/create_sensor_metadata.sql.
3. Зайдите в только что созданную базу данных `sensor_metadata` и убедитесь, что в ней есть следующие три таблицы: `sensor_type`, `locations` и `sensors`.

Вы можете проверить правильность создания базы данных следующим запросом:

```
select
    st.sensor_type as sensorType,
    l.customer as customer,
    l.department as department,
    l.building_name as buildingName,
    l.room as room,
    l.floor as floor,
    l.location_on_floor as locationOnFloor,
    l.latitude,
    l.longitude
from
    sensors s
        inner join
```

```

    sensor_type st ON s.sensor_type_id =
st.sensor_type_id
        inner join
            location l ON s.location_id = l.location_id
where
    s.sensor_id = 1;

```

Результат выполненного запроса должен выглядеть таким образом.

sensorType	customer	department	buildingName	room	floor	locationOnFloor	latitude	longitude
Temperature	Abc Labs	R & D	222 Broadway	101	Floor1	C-101	40.710936	-74.0085

Наша база данных `sensor_metadata` готова для поиска необходимых метаданных датчиков. В следующем разделе мы приступим к созданию контейнера данных в Logstash.

Создание контейнера данных Logstash

Теперь, когда у нас настроен механизм автоматического создания индексов Elasticsearch и базы метаданных, мы можем сфокусироваться на создании контейнера данных в Logstash. Для чего нужен наш контейнер данных? Он должен выполнять следующие действия.

1. Принимать запросы JSON по сети (через HTTP).
2. Дополнять JSON метаданными, которые есть в нашей базе данных MySQL.
3. Сохранять в Elasticsearch полученные в результате документы.

Эти три основные функции, которые нам необходимы, согласовываются с плагинами ввода данных контейнера Logstash,

фильтрации и вывода соответственно. Полную конфигурацию Logstash для этого контейнера данных вы можете найти по следующей ссылке: https://github.com/pranavshukla/learningelasticsearch/tree/master/chapter-10/files/logstash_sensor_data_http.conf.

Теперь мы пошагово рассмотрим, как достичь финальной цели нашего контейнера данных. Начнем с приема запросов JSON по сети (через HTTP).

Прием запросов JSON по сети

Для этой задачи нам понадобится плагин ввода. Logstash поддерживает плагин ввода `http`, который выполняет необходимые действия, а именно создание HTTP-интерфейса, с помощью которого различные типы данных могут попасть в Logstash.

Часть файла `logstash_sensor_data_http.conf`, которая содержит фильтр ввода, выглядит следующим образом:

```
input {
  http {
    id => "sensor_data_http_input"
  }
}
```

В поле `id` находится строка с уникальным идентификатором этого фильтра ввода. Нам не нужно ссылаться на это название в файле; мы просто выберем название `sensor_data_http_input`.

Документация по плагину ввода HTTP доступна по следующей ссылке: <https://www.elastic.co/guide/en/logstash/current/plugins-inputs-http.html>. Поскольку мы используем конфигурацию плагина по умолчанию, то просто указали идентификатор (`id`). Нам необходимо настроить безопасность для этой части HTTP, так как она будет открыта в Интернете для того, чтобы датчики могли

отправлять данные откуда угодно. Мы можем задать имя и пароль (параметры `user` и `password`) для защиты, как показано в коде ниже:

```
input {
  http {
    id => "sensor_data_http_input"
    user => "sensor_data"
    password => "sensor_data"
  }
}
```

Когда мы запускаем Logstash с этим плагином ввода, он запускает HTTP, используя порт 8080. Безопасность обеспечивается базовой аутентификацией с указанием имени пользователя и пароля. Мы можем отправить запрос этому контейнеру в Logstash, воспользовавшись командой `curl`, как показано ниже:

```
curl -XPOST -u sensor_data:sensor_data --header
"Content-Type:
application/json" "http://localhost:8080/" -d
'{"sensor_id":1,"time":1512102540000,"reading":16
.24}'
```

Рассмотрим, как дополнить данные JSON, используя метаданные, которые у нас есть в MySQL.

Дополнение JSON метаданными из нашей базы данных MySQL

Дополнение и другая обработка данных внутри контейнера может выполняться с помощью плагинов фильтрации. Мы создали реляционную базу данных, которая содержит таблицы и позволяет искать данные для дополнения входящих запросов JSON.

В Logstash доступен плагин фильтрации, который можно использовать для поиска по любой реляционной базе данных и

дополнения входящих документов JSON. Рассмотрим раздел плагинов в нашем конфигурационном файле Logstash:

```
filter {
    jdbc_streaming {
        jdbc_driver_library => "/path/to/mysql-
connector-java-5.1.45-bin.jar"
        jdbc_driver_class => "com.mysql.jdbc.Driver"
        jdbc_connection_string =>
"jdbc:mysql://localhost:3306/sensor_metadata"
        jdbc_user => "root"
        jdbc_password => "<password>"
        statement => "select st.sensor_type as
sensorType, l.customer as
customer, l.department as department,
l.building_name as buildingName,
l.room as room, l.floor as floor,
l.location_on_floor as locationOnFloor,
l.latitude, l.longitude from sensors s inner
join sensor_type st on
s.sensor_type_id=st.sensor_type_id inner join
location l on
s.location_id=l.location_id      where
s.sensor_id= :sensor_identifier"
        parameters => { "sensor_identifier" =>
"sensor_id"}
        target => lookupResult
    }
}

mutate {
    rename => {"[lookupResult][0][sensorType]" =>
"sensorType"}
    rename => {"[lookupResult][0][customer]" =>
"customer"}
```

```

        rename => {"[lookupResult][0][department]" =>
"department"}
        rename => {"[lookupResult][0][buildingName]" =>
"buildingName"}
        rename => {"[lookupResult][0][room]" =>
"room"}
        rename => {"[lookupResult][0][floor]" =>
"floor"}
        rename => {"[lookupResult][0][locationOnFloor]" =>
"locationOnFloor"}
    add_field => {
        "location" =>
            "%{lookupResult[0]latitude},%{lookupResult[0]longitude}"
    }
    remove_field => ["lookupResult", "headers",
"host"]
}
}

```

Как вы можете увидеть, в файле используются два плагина фильтрации:

- `jdbc_streaming`;
- `mutate`.

Рассмотрим, для чего они нужны.

Плагин `jdbc_streaming`

Мы явно указали расположение базы данных, к которой хотим подключиться, имя пользователя и пароль, файл `.jar` драйвера JDBC и класс.

Скачайте последнюю версию драйвера MySQL JDBC, также известного под названием *Connector/J*, по следующей ссылке: <https://dev.mysql.com/downloads/connector/j/>. На момент написания книги последней была версия 5.1.45, которая работает с версиями MySQL 5.5, 5.6 и 5.7. Скачайте файл `.tar/.zip`, содержащий драйвер, и распакуйте его в вашей системе. Путь к этому извлеченному файлу `.jar` должен быть указан в параметре `jdbc_driver_library`.

В общем счете вам необходимо проверить и обновить следующие параметры в конфигурации Logstash для указания вашей базы данных и файла драйвера `.jar`:

- `jdbc_connection_string`;
- `jdbc_password`;
- `jdbc_driver_library`.

Параметр `statement` имеет тот же запрос SQL, который мы видели ранее. Он ищет метаданные для выбранного `sensor_id`. Успешный запрос извлечет все дополнительные поля для этого `sensor_id`. Результат запроса сохраняется в новом поле — `lookupResult`, как указано в параметре `target`.

В результате мы должны получить документ следующего вида:

```
{  
  "sensor_id": 1,  
  "time": 1512102540000,  
  "reading": 16.24,  
  "lookupResult": [  
    {  
      "buildingName": "222 Broadway",  
      "sensorType": "Temperature",  
      "latitude": 40.710936,
```

```

        "locationOnFloor": "Desk 102",
        "department": "Engineering",
        "floor": "Floor 1",
        "room": "101",
        "customer": "Linkedin",
        "longitude": -74.0085
    }
],
"@timestamp": "2017-12-07T12:12:37.477Z",
"@version": "1",
"host": "0:0:0:0:0:0:0:1",
"headers": {
    "remote_user": "sensor_data",
    "http_accept": "*/*",
    ...
}
}

```

Как видите, плагин фильтрации `jdbc_streaming` добавил несколько полей, помимо `lookupResult`. Эти поля были добавлены Logstash, а поле `headers` – плагином ввода HTTP.

Далее мы воспользуемся плагином фильтрации `mutate` для модификации этого документа JSON в желаемый конечный результат в Elasticsearch.

Плагин `mutate`

Как вы видели в предыдущем разделе, вывод плагина фильтрации `jdbc_streaming` имеет некоторые нежелательные последствия. Для наших данных JSON необходимы такие изменения:

- перенос найденных полей под `lookupResult` напрямую в JSON;

- комбинирование полей широты и долготы под `lookupResult` в виде поля местоположения;
- удаление ненужных полей.

```
mutate {
    rename => {["lookupResult"][0]["sensorType"] =>
    "sensorType"}
    rename => {["lookupResult"][0]["customer"] =>
    "customer"}
    rename => {["lookupResult"][0]["department"] =>
    "department"}
    rename => {["lookupResult"][0]["buildingName"] =>
    "buildingName"}
    rename => {["lookupResult"][0]["room"] => "room"}
    rename => {["lookupResult"][0]["floor"] =>
    "floor"}
    rename => {["lookupResult"][0]["locationOnFloor"] =>
    "locationOnFloor"}
    add_field => {
        "location" => "%{lookupResult[0]latitude},%{lookupResult[0]longitude}"
    }
    remove_field => ["lookupResult", "headers", "host"]
}
```

Посмотрим, как плагин `mutate` выполнит эти задачи.

Перенос найденных полей под `lookupResult` напрямую в JSON

Как вы уже видели, `lookupResult` находится в массиве с единственным элементом (под индексом 0 в массиве). Нам необходимо перенести все поля в этот элемент массива

непосредственно под данные JSON. Для этого выполним операцию `rename` для каждого поля.

Например, следующая операция переименует существующее поле `sensorType` непосредственно под данными JSON:

```
rename => {"[lookupResult][0][sensorType]" =>  
"sensorType"}
```

Мы выполним такую операцию для всех найденных полей, которые были возвращены запросом SQL.

Комбинирование полей широты и долготы под `lookupResult` в виде поля местоположения

Помните, как мы определяли разметку шаблона нашего индекса? Мы указали, что поле `location` должно иметь тип `geo_point`. Тип `geo_point` принимает значение в строковом формате, а именно объединенные данные широты и долготы, разделенные запятой.

Для получения такого результата предназначена операция `add_field`, создающая поле `location`, как видно в следующем примере:

```
add_field => {  
    "location" => "%{lookupResult[0]latitude},%  
{lookupResult[0]longitude}"  
}
```

На данном этапе у вас должно быть новое поле с названием `location`, добавленное в данные JSON именно так, как требовалось. Далее мы приступим к удалению ненужных полей.

Удаление ненужных полей

После переноса всех элементов из поля `lookupResult` напрямую в JSON это поле нам больше не нужно. Аналогичным образом мы не хотим хранить поля `headers` и `host` в индексе Elasticsearch. Мы удалим их все, используя следующую операцию:

```
remove_field => ["lookupResult", "headers", "host"]
```

Наконец у нас есть данные JSON в той структуре, которую мы хотим видеть в индексе Elasticsearch. Далее узнаем, как отправить эти данные в Elasticsearch.

Сохранение в Elasticsearch полученных в результате документов

Для отправки данных мы задействуем плагин вывода Elasticsearch, который поставляется в комплекте с Logstash. Он крайне прост в использовании; нам нужно только указать `elasticsearch` в тегах вывода:

```
output {  
    elasticsearch {  
        hosts => ["localhost:9200"]  
        index => "sensor_data-%{+YYYY.MM.dd}"  
    }  
}
```

Мы указали параметры `hosts` и `index` для отправки данных в правильный индекс в правильном кластере. Обратите внимание, что название индекса имеет вид `%{+YYYY.MM.dd}`. Название используемого индекса задается согласно текущему времени события и форматируется соответствующим образом.

Помните, что мы определили шаблон индекса `sensor_data*`. При отправке первого события 1 декабря 2017 года указанный плагин вывода отправит событие в индекс `sensor_data-2017.12.01`.

Если вы хотите отправлять события в защищенный кластер Elasticsearch, настройку которого с помощью X-Pack мы рассматривали в главе 8, можете изменить имя пользователя и пароль, как указано ниже:

```
output {  
    elasticsearch {  
        hosts => ["localhost:9200"]  
        index => "sensor_data-%{+YYYY.MM.dd}"  
        user => "elastic"  
        password => "elastic"  
    }  
}
```

Таким образом, у вас будет один индекс на каждый день, в пределах которого будут храниться данные этого дня.

Теперь, когда контейнер данных Logstash готов, отправим данные.

Отправка данных в Logstash через HTTP

На данном этапе датчики уже могут начинать отправлять свои показания в контейнер данных Logstash, который мы создали в предыдущем разделе. Им нужно отправлять данные следующим образом:

```
curl -XPOST -u sensor_data:sensor_data --header  
"Content-Type:  
application/json" "http://localhost:8080/" -d  
'{"sensor_id":1,"time":1512102540000,"reading":16  
.24}'
```

Поскольку у нас нет настоящих датчиков, мы будем симулировать данные путем отправки запросов подобного типа. Данные симуляции и скрипт для их отправки в виде кода доступны

по следующей ссылке: <https://github.com/pranavshukla/learningelasticstack/tree/master/chapter-10/data>.

Если вы используете ОС Linux или macOS, откройте терминал и измените папку на ваше рабочее пространство Elastic Stack, которое получили с GitHub.



Если вы используете Windows, вам понадобится Linux-подобная оболочка, которая поддерживает команду curl и базовые команды BASH (Bourne Again SHell). Если у вас уже есть рабочее пространство GitHub, вы можете использовать Git for Windows, в котором есть Git BASH. Его вы можете задействовать для выполнения скрипта, который загружает данные. При отсутствии этого компонента скачайте и установите Git for Windows по следующей ссылке: <https://git-scm.com/download/win>. После чего запустите Git BASH для выполнения команд, приведенных в этой главе.

Теперь перейдем в каталог chapter-10/data и выполним load_sensor_data.sh:

```
$ pwd  
/Users/pranavshukla/workspace/learningelasticstack  
$ cd chapter-10/data  
$ ls  
load_sensor_data.sh sensor_data.json  
$ ./load_sensor_data.sh
```

Скрипт load_sensor_data.sh читает документ sensor_data.json строка за строкой и передает данные в Logstash, используя команду curl.

Мы только что воспроизвели в Logstash показания датчиков за один день — они считывались каждую минуту из различных географических мест. Контейнер данных Logstash, который мы создали ранее, должен дополнить данные и отправить их в Elasticsearch.

Теперь настало время перейти в Kibana и получить детальные отчеты по данным.

Визуализация данных в Kibana

Мы успешно настроили контейнер данных Logstash, а также загрузили с его помощью данные в Elasticsearch. Теперь перейдем к исследованию данных и созданию панели управления, которая поможет нам получить детальную картину.

Начнем с проверки корректности загрузки данных. Для этого перейдем на страницу **Dev Tools** (Инструменты разработчика) в Kibana и выполним следующий запрос:

```
GET /sensor_data-*/_search?size=0
{
  "query": {"match_all": {}}
}
```

Запрос запустит поиск данных по всем индексам, которые совпадают с шаблоном `sensor_data-*`. Мы должны получить большое количество записей в индексе, если данные корректно проиндексированы.

Мы рассмотрим следующие темы.

- Настройка шаблона индекса в Kibana.
- Создание визуализаций.
- Создание панели управления с помощью визуализаций.

Настройка шаблона индекса в Kibana

Прежде чем мы сможем приступить к созданию визуализаций, необходимо настроить шаблон индекса для всех индексов, которые будут потенциально использоваться приложением аналитики данных датчиков. Мы это делаем потому, что наши названия индексов меняются динамически. Каждый день у нас будет появляться новый индекс, нам необходимо иметь возможность создавать визуализации и панели управления, которые смогут работать с несколькими индексами данных датчиков. Для этого перейдите на вкладку **Management** (Управление) в Kibana и щелкните на ссылке **Index Patterns** (Шаблоны индексов) (рис. 10.2).

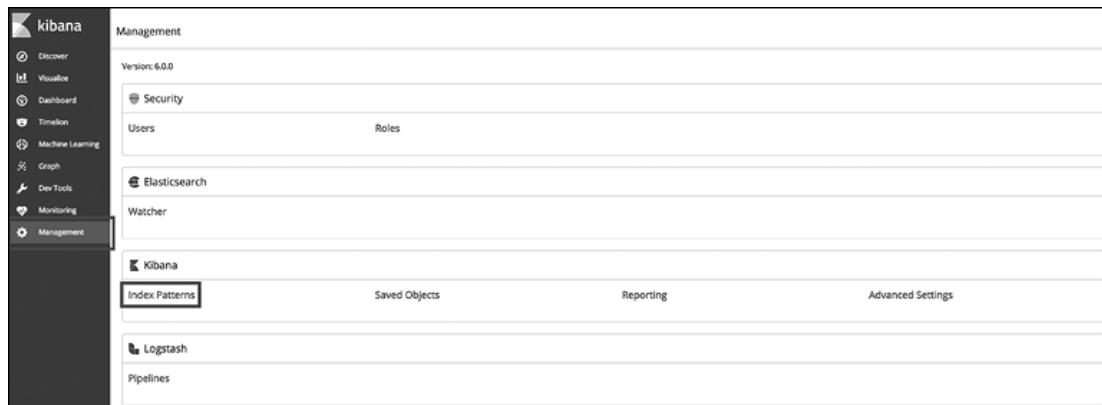


Рис. 10.2

Нажмите кнопку **Create Index Pattern** (Создать шаблон индекса) и добавьте `sensor_data*`, как показано на рис. 10.3. В поле **Time Filter field name** (Название поля фильтра времени) укажите нужное поле и нажмите кнопку **Create** (Создать).



Рис. 10.3

Мы успешно создали шаблон индекса для наших данных с датчиков. Далее приступим к созданию визуализаций.

Создание визуализаций

Наверняка на текущий момент у вас уже накопились вопросы о данных, ответы на которые можно быстро получить с помощью визуализаций. Кроме того, можно сохранить эти визуализации на панели управления для последующего использования при необходимости. Начнем со списка вопросов и постараемся создать визуализации, которые предоставляют ответы на них.

Попробуем найти ответы на следующие вопросы.

- Как меняется средняя температура со временем?
- Как меняется средняя влажность со временем?
- Как меняется температура и влажность в каждом месте со временем?
- Могу ли я визуализировать температуру и влажность на карте?
- Как датчики распределены по отделам?

Для получения ответов создадим визуализации и начнем с первого вопроса.

Как меняется средняя температура со временем?

В данном случае нам потребуется агрегация статистики. Нам необходимо знать среднюю температуру по всем датчикам температуры, независимо от их местонахождения или других критериев. Как вы видели в главе 7, следует перейти на вкладку

Visualize (Визуализации), после чего создать новую визуализацию нажатием кнопки **+**.

Выберите **Basic Charts** (Базовые диаграммы), далее укажите **Line** (Линейная). На следующей странице для настройки линейной диаграммы выполните шаги 1–5 (рис. 10.4).

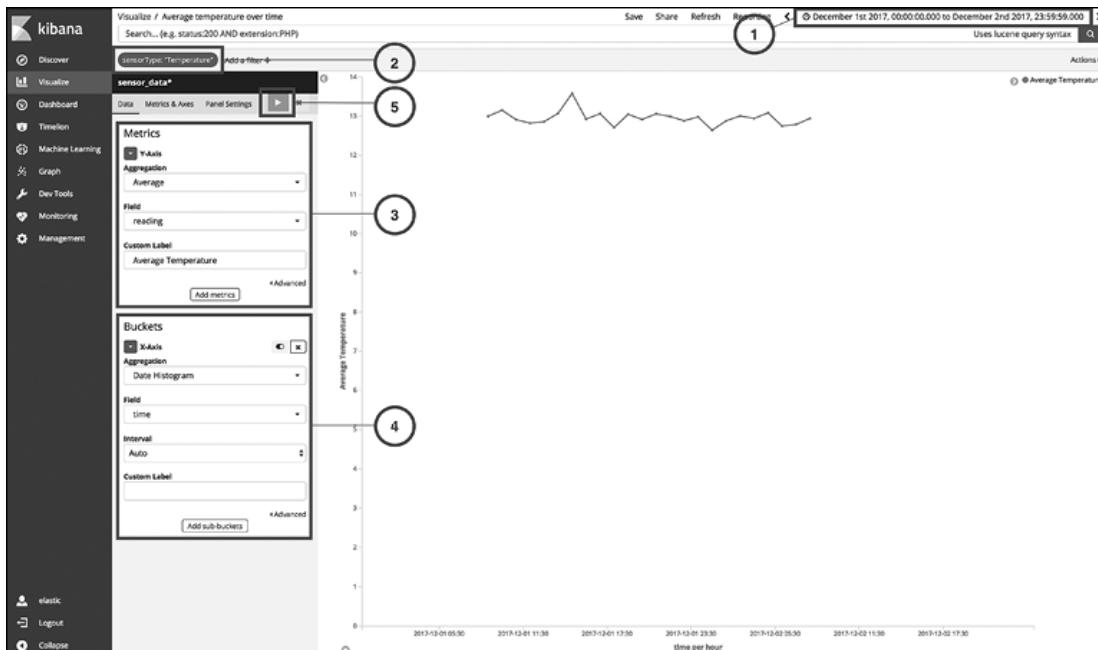


Рис. 10.4

1. Щелкните на маленьком значке с часами в верхнем правом углу, выберите **Absolute** (Абсолютный) и укажите диапазон с 1 по 2 декабря 2017 года. Это необходимо сделать, так как наши симулированные данные датируются 1 декабря 2017 года.
2. Щелкните на ссылке **Add a filter** (Добавить фильтр); выберите фильтр следующим образом: **sensorType is Temperature**. Нажмите кнопку **Save** (Сохранить). У нас есть два типа датчиков, температуры и влажности. Для текущей визуализации нам нужны только показания температуры. Поэтому мы выбрали этот фильтр.

3. Из раздела **Metrics** (Метрики) выберите нужные значения. Нас интересует среднее значение показаний. Мы также изменили метку на **Average Temperature** (Средняя температура).
4. Из раздела **Buckets** (Сегменты) выберите агрегацию **Date Histogram** (Гистограмма даты) и поле времени, остальные настройки оставьте как есть.
5. Нажмите треугольную кнопку **Apply changes** (Применить изменения).

Как меняется средняя влажность со временем?

Этот вопрос очень похож на предыдущий. Для ответа на него мы можем повторно использовать предыдущую визуализацию, слегка модифицировав ее, и создать еще одну копию. Начнем с открытия первой визуализации, которую мы сохранили под названием **Average temperature over time** (Средняя температура по времени).

Для обновления визуализации выполните следующие шаги (рис. 10.5).

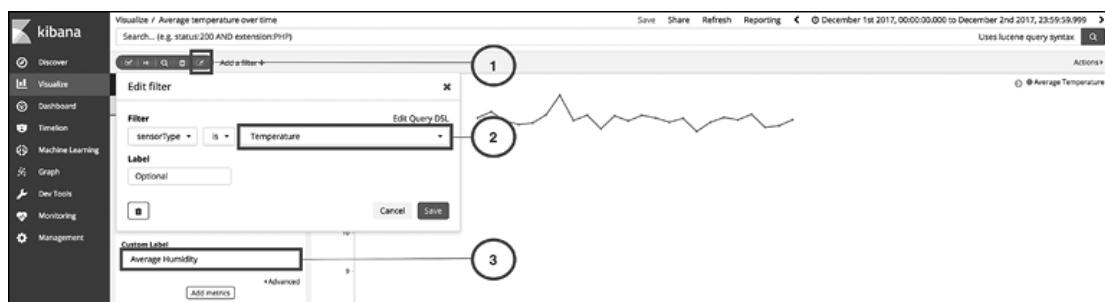


Рис. 10.5

1. Наведите указатель мыши на фильтр с меткой **sensorType**: **"Temperature"** и щелкните на значке **Edit** (Редактировать).
2. Измените значение фильтра с **Temperature** (Температура) на **Humidity** (Влажность) и нажмите кнопку **Save** (Сохранить).

3. Измените **Custom Label** (Метка) со значения **Average Temperature** (Средняя температура) на **Average Humidity** (Средняя влажность) и нажмите кнопку **Apply changes** (Применить изменения).

Как вы увидите, диаграмма обновится для датчиков влажности. Щелкните на ссылке **Save** (Сохранить) вверху навигационной панели, запишите новое название визуализации как **Average humidity over time** (Средняя влажность по времени), установите флажок **Save as a new visualization** (Сохранить как новую визуализацию) и нажмите кнопку **Save** (Сохранить). Так вы создадите вторую визуализацию и получите ответ на второй вопрос.

Как меняется температура и влажность в каждом месте со временем?

На этот раз нам нужно больше деталей, чем в первых двух вопросах. Мы хотим узнать, как меняются температура и влажность в каждом месте со временем. Создадим решение для температуры.

Перейдите на вкладку **Visualizations** (Визуализации) в Kibana и создайте новую линейную диаграмму точно так же, как делали ранее (рис. 10.6).

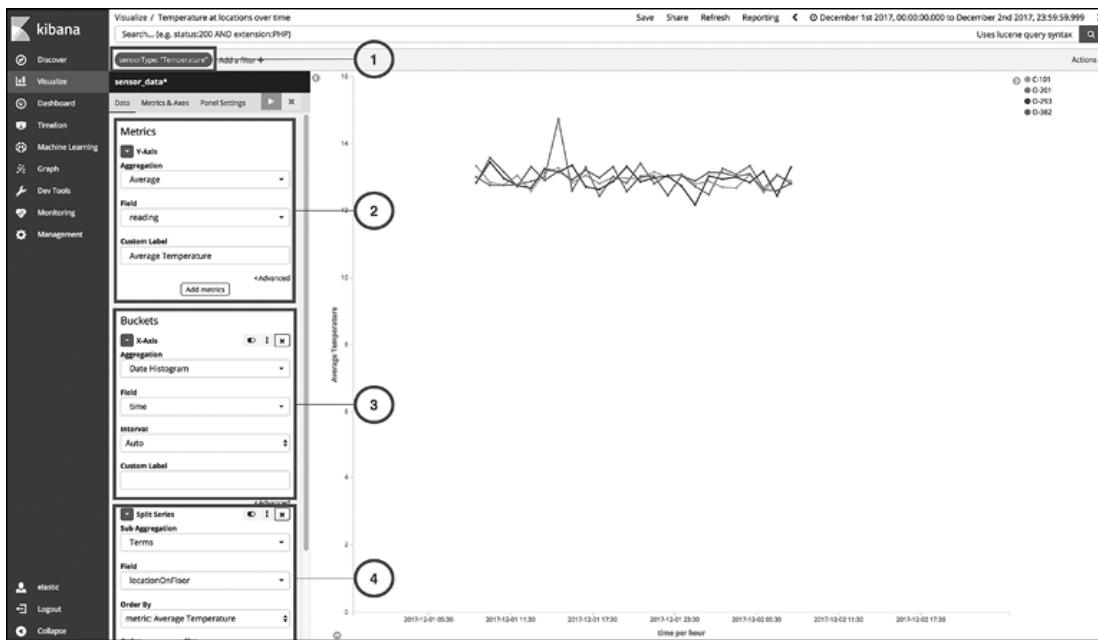


Рис. 10.6

1. Добавьте фильтр для **sensorType: "Temperature"**.
2. Как показано на рис. 10.6, задайте в разделе **Metrics** (Метрики) выполнение агрегации среднего числа для поля **reading**.
3. Поскольку мы выполняем агрегацию данных по полю **time**, необходимо выбрать агрегацию **Date Histogram** (Гистограмма даты) в разделе **Buckets** (Сегменты). Здесь следует выбрать поле и оставить **Interval** (Интервал) в положении **Auto** (Автоматически).
4. На этот момент новая визуализация аналогична предыдущей: **Average temperature over time** (Средняя температура по времени). Но нам недостаточно видеть только среднюю температуру, нам надо видеть ее по значению **locationOnFloor**, которое является нашей единицей идентификации местонахождения. Поэтому в этом шаге мы разбиваем ряд с помощью агрегации условий по полю **locationOnFloor**. Мы хотим выполнить сортировку по **metric: Average Temperature** и установить в

качестве размера сортировки значение 5, чтобы видеть только топ-5 местонахождений.

Теперь мы создали визуализацию, которая показывает изменение температуры для каждого значения locationOnFloor в наших данных. Вы можете отчетливо видеть перепад **0-201** 1 декабря 2017 года в 15:00 IST. Из-за этого перепада мы получили перепад средней температуры в это же время в нашей первой визуализации. Это важная информация, которая появилась благодаря графику.

Визуализация для влажности может быть создана путем повторения всех шагов, только с заменой температуры влажностью.

Могу ли я визуализировать температуру и влажность на карте?

Мы можем визуализировать температуру и влажность на карте, используя визуализацию в виде карты координат. Создайте новую визуализацию типа **Coordinate Map** (Карта координат) и выполните следующие шаги (рис. 10.7).



Рис. 10.7

1. Как и в предыдущих визуализациях, добавьте фильтр для **sensorType: "Temperature"**.
2. В разделе **Metrics** (Метрики) выберите агрегацию среднего числа (**Average**) для поля **reading**, как уже было сделано ранее.
3. Поскольку мы используем карту координат, необходимо выбрать агрегацию сетки GeoHash и указать поле **geo_point**, которое присутствует в наших данных; поле для агрегации — **location**.

Как видите, теперь наши данные графически отображаются на карте. Мы можем мгновенно узнать среднюю температуру на каждой площадке при наведении указателя мыши на каждую точку. Сфокусируйте внимание на необходимой части карты и сохраните визуализацию с названием **Temperature over locations** (Температура по местам).

Аналогичным образом можно создать визуализацию в виде карты координат для датчиков влажности.

Как датчики распределены по отделам?

Что, если мы хотим увидеть, как датчики распределены по разным отделам? Помните, что в наших данных есть поле **department**, которое мы получили после дополнения данных на основе **sensor_id**. Для визуализации распределения данных по различным значениям полей типа **keyword**, таким как **department**, идеально подойдут круговые диаграммы. Создадим новую визуализацию типа **Pie** (Круговая диаграмма).

Проследуйте по шагам, отмеченным на рис. 10.8.

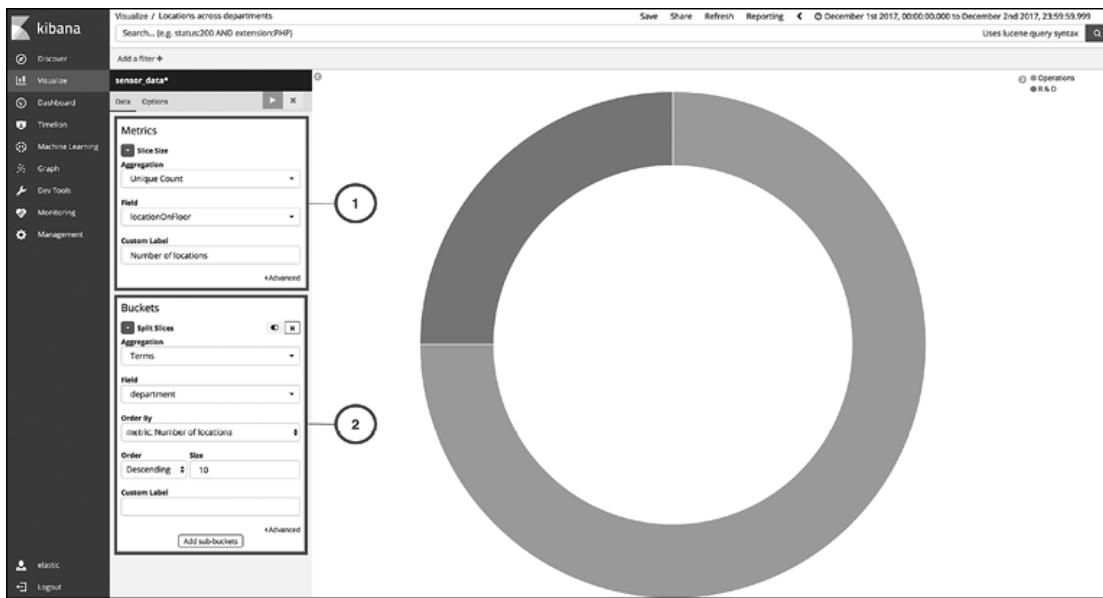


Рис. 10.8

1. В разделе **Metrics** (Метрики) выберите агрегацию **Unique Count** (Уникальный подсчет) и поле `locationOnFloor`. Вы можете отредактировать значение **Custom Label** (Пользовательская метка) согласно количеству мест.
2. В разделе **Buckets** (Сегменты) необходимо выбрать агрегацию терминов по полю `department`, если хотим собрать данные по разным отделам.

Нажмите кнопку **Apply changes** (Применить изменения) и сохраните визуализацию как **Locations across departments** (Места по отделам). Аналогичным образом вы можете создать визуализацию для демонстрации мест по разным зданиям. Ее можно назвать **Locations across buildings** (Места по зданиям). Это поможет нам узнать, в каких местах установлено наблюдение в каждом здании.

Далее мы создадим панель управления для сбора всех визуализаций вместе.

Создание панели управления

Панель управления позволяет организовать несколько визуализаций воедино, сохранять их, а также делиться ими с другими людьми. В возможности наблюдать за несколькими визуализациями одновременно есть свои преимущества. Вы можете фильтровать данные по различным критериям и увидеть отфильтрованный результат на всех визуализациях. Благодаря такому функционалу вы можете собрать больше сведений о своих данных либо получить ответы на самые сложные вопросы.

Создадим панель управления из уже созданных визуализаций. Щелкните на вкладке **Dashboard** (Панель управления) на навигационной панели слева в Kibana. Нажмите кнопку **+** для создания новой панели управления.

Нажмите кнопку **Add** (Добавить) в меню вверху для добавления визуализаций на вашу новую панель управления. Так вы получите полный список уже созданных визуализаций в раскрывающемся меню. Вы можете добавить визуализации одну за другой, а также менять их местами и редактировать размер для создания панели управления, которая идеально соответствует вашим требованиям.

Взглянем на то, какой может быть панель управления для приложения, которое мы создаем (рис. 10.9).

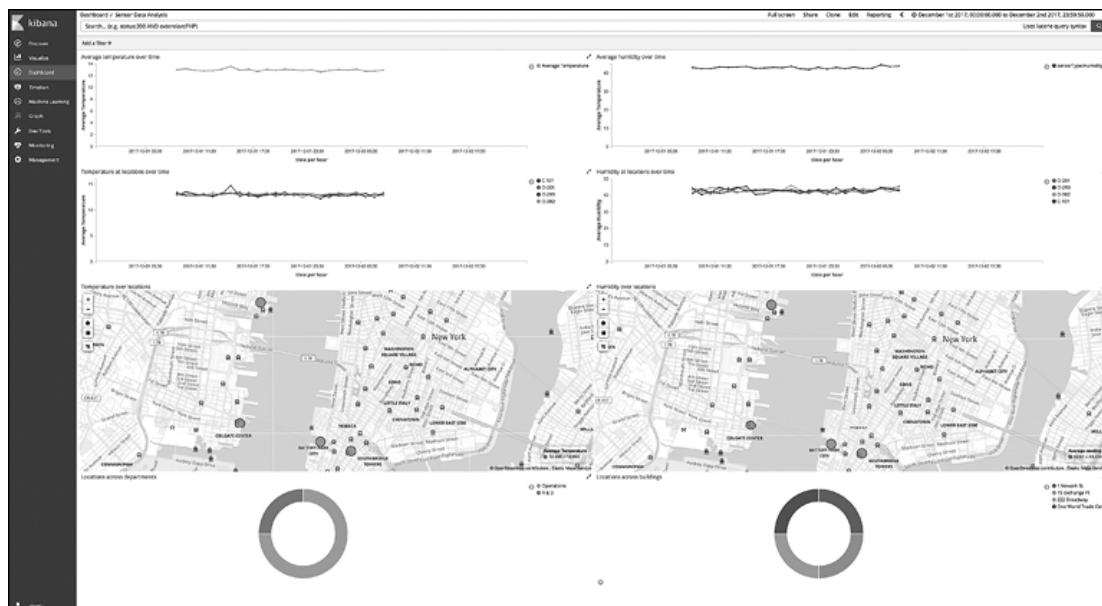


Рис. 10.9

На панель управления можно добавить фильтры — для этого щелкните на ссылке **Add a filter** (Добавить фильтр) в верхнем левом углу панели управления. Выбранный фильтр будет применен ко всем диаграммам.

Визуализации интерактивны; например, щелкнув на одной из частей круговой диаграммы, вы можете применить этот фильтр глобально. Рассмотрим, в каких случаях это может пригодиться.

Если вы щелкнете на сегменте диаграммы для здания 222 Бродвея в нижнем правом углу, то увидите, что в фильтры был добавлен `buildingName: "222 Broadway"`. Таким образом, вы можете видеть все данные из перспективы всех датчиков этого здания (рис. 10.10).

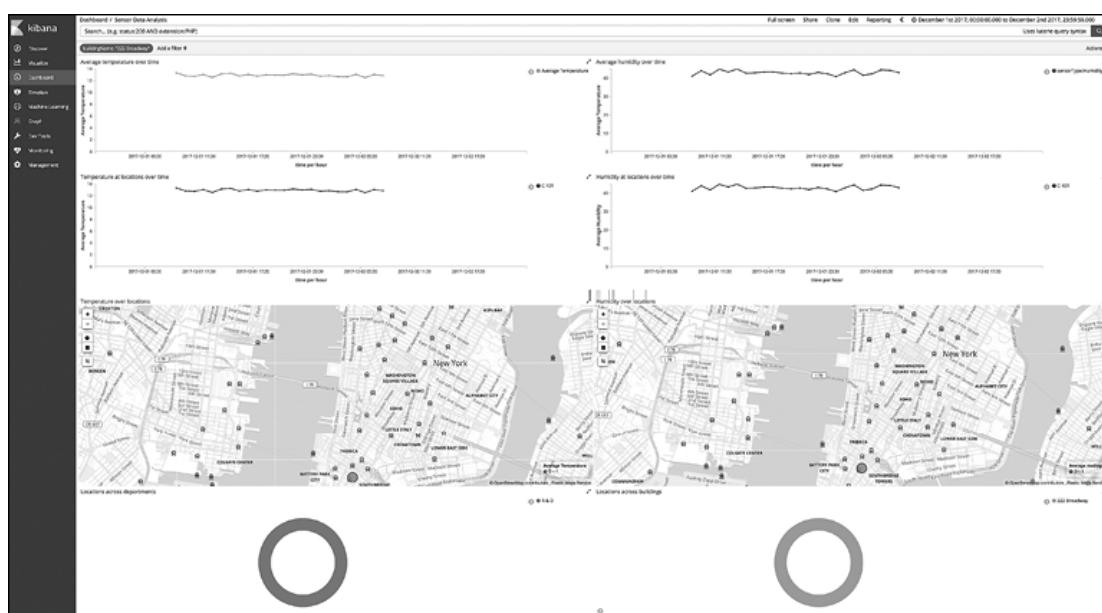


Рис. 10.10

Удалим этот фильтр. Для этого наведите указатель мыши на фильтр `buildingName: "222 Broadway"` и щелкните на значке с урной. Далее мы попробуем поработать с одной из линейных диаграмм, то есть с визуализацией **Temperature at locations over time** (Температура по местам во времени).

Как мы уже наблюдали, 1 декабря 2017 года был перепад показателей. Вы можете увеличить детализацию по определенному периоду времени, если нарисуете мышью прямоугольник на той части диаграммы, которую хотите увеличить. Иными словами, обведите перепад прямоугольником, удерживая нажатой левую кнопку мыши. В результате изменится фильтр времени, который изначально был применен ко всей панели управления (отображается в правом верхнем углу).

Посмотрим, удастся ли нам тщательнее исследовать данные после этой операции фокусировки на периоде времени (рис. 10.11).

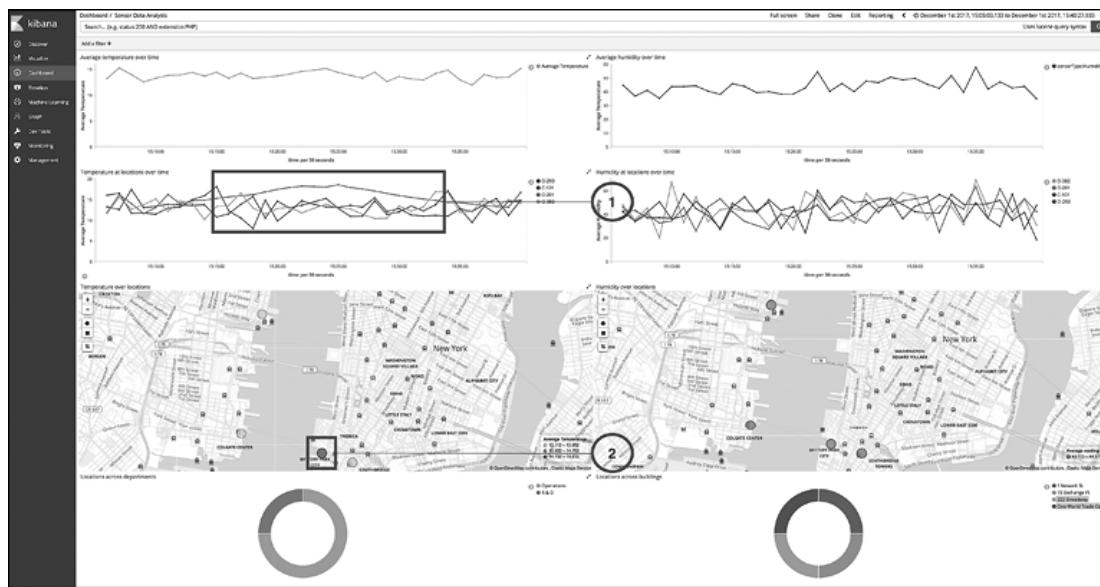


Рис. 10.11

Мы получили следующие факты.

- Температура, согласно датчику на площадке О-201, стабильно поднимается на протяжении этого времени.
- На карте координат вы можете увидеть, что выделенный круг темнее, по сравнению с другими местами, которые выделены более светлым оттенком (в оригинале красный и желтый цвета соответственно. — Примеч. ред.). Это указывает на то, что

данная площадка имеет ненормально высокую температуру на фоне других.

Как видите, интерактивное использование диаграмм и выбор различных фильтров дают возможность получить необходимые сведения о ваших данных.

Таким образом, мы продемонстрировали возможности созданного приложения, используя компоненты Elastic Stack.

Резюме

В этой главе мы создали приложение аналитики данных с датчиков, которое имеет широкий спектр применения и относится к развивающейся отрасли Интернета вещей. Мы создали приложение для аналитики, используя только компоненты Elastic Stack, не прибегая к каким-либо другим инструментам или языкам программирования, и получили мощный инструмент, который позволяет обрабатывать большие объемы данных.

Мы начали с самых азов, а именно — с моделирования данных для Elasticsearch. Далее мы разработали контейнер данных, который имеет защиту и может получать данные из Интернета посредством HTTP. Мы дополнили входящие данные с помощью метаданных, которые расположены в реляционной БД и хранятся в Elasticsearch. Мы отправили тестовые данные по HTTP точно так же, как настоящие датчики отправляют данные через Интернет. Мы создали визуализации, которые могут дать ответы на некоторые типичные вопросы о данных. После чего мы собрали все визуализации на одной мощной интерактивной панели управления.

В главе 11 мы создадим еще одно приложение для решения реальных задач с помощью Elastic Stack.

11. Мониторинг серверной инфраструктуры

Из предыдущей главы вы узнали, как эффективно запускать Elastic Stack в рабочей среде и каким правилам рекомендуется следовать при работе с Elastic Stack.

В этой главе рассказывается, как использовать платформу Beats для мониторинга серверной инфраструктуры. Мы детально рассмотрим Metricbeat — компонент Beats, который помогает ИТ-администраторам и командам поддержки мониторить свои приложения и серверную инфраструктуру, а также оперативно реагировать в случае неполадок или недоступности сервера.

В этой главе мы разберем следующие темы.

- Компонент Beats — Metricbeat, который используется для сбора метрик в системах и приложениях.
- Установка и настройка Metricbeat.
- Архитектура внедрения.

Metricbeat

Metricbeat — это легковесный поставщик для периодического сбора метрик в операционных системах и сервисах, запущенных на ваших серверах. Он помогает мониторить систему путем сбора метрик из нее, а также таких сервисов, как Apache, MongoDB, Redis и пр., запущенных на выбранном сервере. Metricbeat может передавать собранные метрики напрямую в Elasticsearch или отправлять в Logstash, Redis или Kafka. Для мониторинга сервисов Metricbeat установлен на пограничный сервер; также есть возможность сбора метрик и с

удаленного сервера. Тем не менее рекомендуется устанавливать Metricbeat на тех серверах, на которых запущены сервисы.

Скачивание и установка Metricbeat

Перейдите по ссылке <https://www.elastic.co/downloads/beats/metricbeat> и скачайте файл ZIP/TAR в зависимости от вашей операционной системы, как показано на рис. 11.1. Установка Metricbeat проста и понятна.

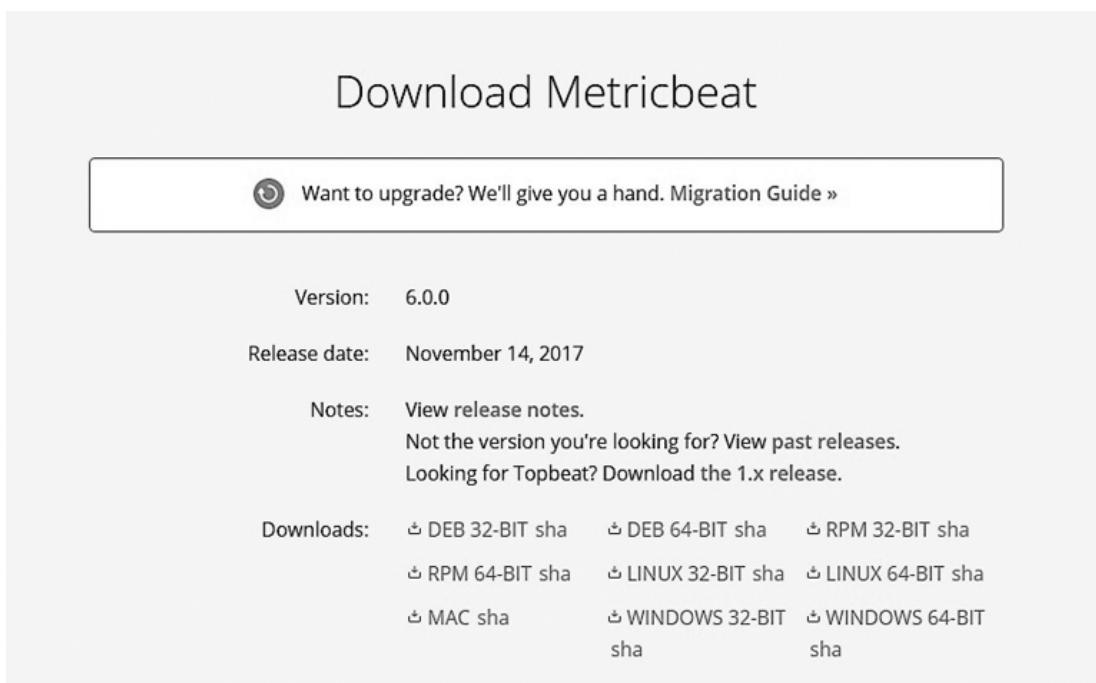


Рис. 11.1



Версии Beats 6.0.x совместимы с Elasticsearch 5.6.x и 6.0.x и Logstash версий 5.6.x и 6.0.x. Полную схему совместимости вы можете найти по ссылке

https://www.elastic.co/support/matrix#matrix_compatibility.

Прежде чем рассматривать примеры использования Elasticsearch и Logstash совместно с Beats в этой главе, убедитесь, что у вас установлены совместимые версии.

Установка в Windows

Распакуйте скачанный файл. После распаковки перейдите в созданную папку, как показано в следующем фрагменте кода:

```
D:>cd D:\packt\metricbeat-6.0.0-windows-x86_64
```

Для установки Filebeat как службы Windows выполните следующие шаги.

1. Откройте Windows PowerShell с правами администратора и перейдите в папку с распакованными файлами.
2. Из командной строки PowerShell выполните следующие команды для установки Metricbeat как службы Windows:

```
PS >cd D:\packt\metricbeat-6.0.0-windows-x86_64  
PS D:\packt\metricbeat-6.0.0-windows-x86_64>.\installservice-metricbeat.ps1
```



Если в вашей системе отключено выполнение скриптов, вам необходимо настроить нужные политики для текущей сессии

таким образом, чтобы вышеописанный скрипт был успешно выполнен. Например:

```
PowerShell.exe -ExecutionPolicy Unrestricted -File .\install-service-metricbeat.ps1
```

Установка в Linux

Распакуйте пакет tar.gz и перейдите в созданную папку, как показано ниже:

```
$> tar -xzf metricbeat-6.0.0-linux-x86_64.tar.gz  
$>cd metricbeat
```



Для установки с использованием deb или rpm выполните соответствующие команды в терминале:

deb:

```
curl -L -O  
https://artifacts.elastic.co/downloads/beats/metricbeat/metricbeat-6.0.0-amd64.deb  
sudo dpkg -i metricbeat-6.0.0-amd64.deb
```

rpm:

```
curl -L -O  
https://artifacts.elastic.co/downloads/beats/metricbeat/metricbeat-6.0.0-amd64.rpm
```

```
metricbeat-6.0.0-x86_64.rpm  
sudo rpm -vi metricbeat-6.0.0-x86_64.rpm
```

Metricbeat будет установлен в папку /usr/share/metricbeat. Файлы конфигурации расположены в /etc/metricbeat. Скрипт init будет находиться в папке /etc/init.d/metricbeat. Файлы log – в папке /var/log/metricbeat.

Архитектура

Metricbeat состоит из двух компонентов: *модулей* и *метрик-сетов*. Модуль Metricbeat определяет логику сбора данных из определенных сервисов, таких как MongoDB, Apache и т.д. В модуле указываются сведения о сервисах, в том числе способ подключения, частота сбора метрик, типы метрик.

Каждый модуль имеет один или несколько метрик-сетов. Метрик-сет – это компонент, который собирает список связанных метрик по службам операционной системы, используя один запрос. Он структурирует данные событий и отправляет их в настроенный вывод, например в Elasticsearch или Logstash.

Metricbeat собирает метрики периодически, в зависимости от интервала, указанного в конфигурационном файле `metricbeat.yml`, и отправляет события в настроенный вывод. События отправляются асинхронно, по аналогии с работой модуля Filebeat, который гарантирует хотя бы одну доставку; если настроенный вывод недоступен, то события будут утрачены.

Например, модуль MongoDB предоставляет метрик-сеты `status` и `dbstats`, которые собирают информацию и статистику путем обработки возвращенного ответа, полученного из выполняемых команд `db.serverStatus()` и `db.stats()` в MongoDB, как показано на рис. 11.2.

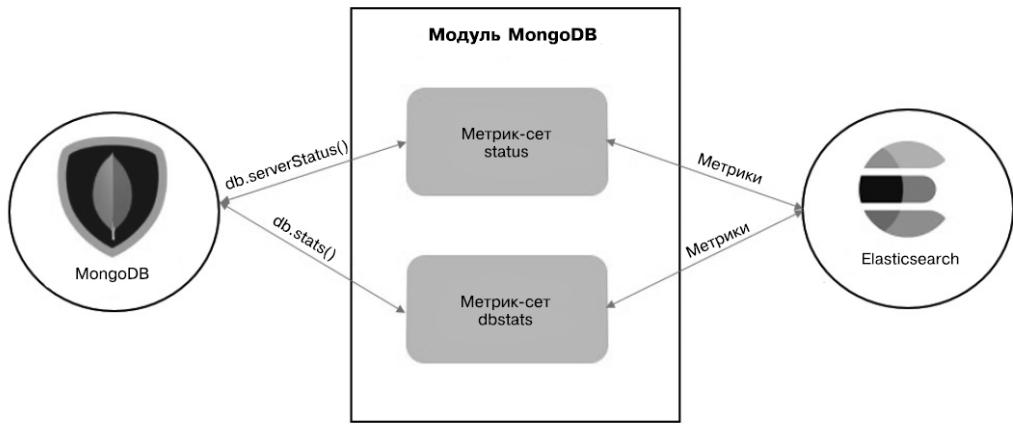


Рис. 11.2

Ниже перечислены основные преимущества Metricbeat.

- **Metricbeat отправляет и события ошибок.** Если сервис недоступен или вышел из строя, Metricbeat отправит события с полными описаниями ошибок, полученными из системы. Это полезно для отладки или поиска причины недоступности сервиса.
- **Совмещает связанные метрики в единое событие.** Metricbeat получает все связанные метрики из системы с помощью одного запроса вместо того, чтобы делать множественные запросы для получения каждой метрики отдельно. Благодаря этому снижается нагрузка на сервисы/системы. Полученные метрики комбинируются в единое событие и отправляются в настроенный вывод.
- **Отправляет информацию метаданных.** Отправляемая Metricbeat метрика содержит и числа, и строки с информацией статуса. Кроме того, доставляются основные метаданные о каждой метрике как часть события. Это

полезно для разметки соответствующих типов данных во время хранения и помогает с запросами/фильтрацией данных, идентификации событий по информации метаданных и пр.

- **Отправляет необработанные данные как есть.** Metricbeat отправляет полученные необработанные данные без выполнения какой-либо обработки или операций агрегации, тем самым уменьшается сложность компонента Metricbeat.

Структура событий. Metricbeat отправляет два типа событий:

- обычные события, содержащие полученную метрику;
- события, содержащие ошибки в случаях, когда сервис недоступен.

Все события имеют одинаковую базовую структуру и содержат как минимум следующие поля независимо от типа рабочего модуля:

- `@timestamp` — время записи события;
- `beat.hostname` — имя хоста сервера, на котором работает компонент Beat;
- `beat.name` — название, присвоенное компоненту Beat (по умолчанию совпадает с именем хоста);
- `beat.version` — версия компонента Beat;

- `metricset.module` — название модуля, который отправил данные;
- `metricset.name` — название метрик-сета, который отправил данные;
- `metricset.rtt` — время выполнения запроса в миллисекундах;
- `@metadata.beat` — тип компонента Beat (то есть Metricbeat);
- `@metadata.type` — по умолчанию `doc`;
- `@metadata.version` — версия компонента Beat.

В случае событий с ошибками к событию будет добавлено поле ошибки, такое как `error.message`, `error.code` и `error.type`, содержащее сообщение об ошибке, код и тип.

Типичное событие выглядит следующим образом:

```
{
  "@timestamp": "2017-11-25T11:48:33.269Z",
  "@metadata": {
    "beat": "metricbeat",
    "type": "doc",
    "version": "6.0.0"
  },
  "system": {
    "fsstat": {
      "total_size": {
        "free": 189415194624,
        "used": 189415194624
      }
    }
  }
}
```

```
        "used": 305321828352,
        "total": 494737022976
    },
    "count": 2,
    "total_files": 0
}
},
"metricset": {
    "name": "fsstat",
    "rtt": 2000,
    "module": "system"
},
"beat": {
    "version": "6.0.0",
    "name": "SHMN-IN",
    "hostname": "SHMN-IN"
}
}
```

Пример события ошибки, если модуль MongoDB недоступен, выглядит таким образом:

```
{
    "@timestamp": "2017-11-25T11:53:08.056Z",
    "@metadata": {
        "beat": "metricbeat",
        "type": "doc",
        "version": "6.0.0"
    },
    "metricset": {
        "host": "localhost:27017",
        "name": "fsstat",
        "rtt": 2000,
        "status": "error",
        "time": "2017-11-25T11:53:08.056Z",
        "type": "system"
    }
}
```

```
"rtt": 1003057,  
"module": "mongodb",  
"name": "status"  
},  
"error": {  
    "message": "no reachable servers"  
},  
"mongodb": {  
    "status": {}  
}  
}
```

Вместе с минимальным набором полей (базовая структура события), отправляемым Metricbeat, также могут отправляться поля модулей, в зависимости от включенных модулей. Полный список доставляемых полей по модулям доступен по следующей ссылке:

<https://www.elastic.co/guide/en/beats/metricbeat/current/exported-fields.html>.

Настройка Metricbeat

Конфигурация, связанная с Metricbeat, хранится в конфигурационном файле под названием `metricbeat.yml`, который использует синтаксис YAML.

Файл `metricbeat.yml` содержит следующую информацию:

- конфигурацию модуля;
- общие настройки;
- настройки вывода;

- настройки обработки;
- настройки пути;
- настройки панели управления;
- настройки сбора данных.

Рассмотрим некоторые из этих сведений.



Файл `metricbeat.yml` будет присутствовать в папке установки только в случае, если были использованы файлы `.zip` или `.tar`. Если для установки были использованы файлы `DEB` или `RPM`, файл будет находиться в папке `/etc/metricbeat`.

Конфигурация модуля

Metricbeat поставляется в комплекте с различными модулями. Они призваны собирать метрики из системы и приложений, таких как Apache, MongoDB, Redis, MySQL и др.

Metricbeat предоставляет два способа активизации модулей и метрик-сетов:

- активизацию конфигураций модулей в папке `modules.d`;
- активизацию конфигураций модулей в файле `metricbeat.yml`.

Активизация конфигураций модулей в папке `modules.d`

Папка `modules.d` содержит конфигурации по умолчанию для всех модулей Metricbeat. Конфигурация каждого модуля хранится в файле `.yml`, а название файла соответствует названию модуля. Например, конфигурация модуля MySQL будет храниться в файле `mysql.yml`. По умолчанию, кроме модуля `system`, все другие модули деактивизированы. Для получения списка модулей, доступных в Metricbeat, выполните следующую команду:

Windows :

```
D:\packt\metricbeat-6.0.0-windows-x86_64>metricbeat.exe modules list
```

Linux :

```
[locationOfMetricBeat]$ ./metricbeat modules list
```

Команда `modules list` показывает все доступные модули, а также демонстрирует список активных/неактивных на данный момент модулей.



Если модуль неактивен, то в папке `modules.d` файл конфигурации данного модуля будет иметь расширение `.disabled`.

Базовая конфигурация модуля `mongodb` выглядит следующим образом:

```
- module: mongodb
  metricsets: ["dbstats", "status"]
  period: 10s
```

```
hosts: ["localhost:27017"]  
username: user  
password: pass
```

Для активизации выполните команду `modules enable` с указанием одного или нескольких названий модулей. Например:

Windows :

```
D:\packt\metricbeat-6.0.0-windows-  
x86_64>metricbeat.exe modules enable  
redis mongodb
```

Linux :

```
[locationOfMetricBeat]$ ./metricbeat modules  
enable redis mongodb
```

Аналогичным образом для деактивизации модулей выполните команду `modules disable` с указанием одного или нескольких названий модулей. Например:

Windows :

```
D:\packt\metricbeat-6.0.0-windows-  
x86_64>metricbeat.exe modules disable  
redis mongodb
```

Linux :

```
[locationOfMetricBeat]$ ./metricbeat modules  
disable redis mongodb
```



Для включения динамической перезагрузки конфигурации укажите значение true для параметра reload.enabled. Для указания частоты проверки изменений конфигурации настройте параметр reload.period со значением metricbeat.config.modules.

Например:

```
#metricbeat.yml

metricbeat.config.modules:
  path: ${path.config}/modules.d/*.yml
  reload.enabled: true
  reload.period: 20s
```

Активизация конфигураций модулей в файле metricbeat.yml

Если используются более ранние версии Metricbeat, можно активизировать модули и метрик-сеты напрямую в файле metricbeat.yml, добавив пункты в список metricbeat.modules. Каждый пункт списка начинается с дефиса (-) и состоит из настроек этого модуля. Например:

```
metricbeat.modules:
  -----
  - module: memcached
    metricsets: ["stats"]
    period: 10s
    hosts: ["localhost:11211"]
  -----
  - module: mongodb
```

```
metricsets: ["dbstats", "status"]  
period: 5s
```



Можно указать модуль несколько раз с различными периодами использования для одного или нескольких метрик-сетов. Например:

```
#----- Couchbase Module -----  
-----  
- module: couchbase  
metricsets: ["bucket"]  
period: 15s  
hosts: ["localhost:8091"]  
  
- module: couchbase  
metricsets: ["cluster", "node"]  
period: 30s  
hosts: ["localhost:8091"]
```

Общие настройки

Этот раздел содержит параметры конфигурации и некоторые общие настройки, контролирующие поведение Metricbeat.

Рассмотрим некоторые из этих параметров/настроек.

- * `name` — название отправителя, который публикует сетевые данные. По умолчанию для этого поля указано `hostname`:

```
name: "dc1-host1"
```

- * `tags` — список тегов, которые будут включены в поле `tags` для каждого события, доставляемого с помощью Metricbeat. Благодаря тегам можно легко группировать серверы по различным логическим свойствам, а также легко фильтровать события в Kibana и Logstash:

```
tags: ["staging", "web-tier", "dc1"]
```

- * `max_procs` — максимальное количество ЦП, которые могут быть использованы одновременно. По умолчанию задействуются все логические ЦП, доступные в системе:

```
max_procs: 2
```

Конфигурация вывода

Этот раздел используется для настройки выводов, в которые должны доставляться события. Возможна отправка в один или несколько выводов одновременно. Доступные источники вывода: Elasticsearch, Logstash, Redis, Kafka, файл или консоль.

Некоторые из возможных источников вывода могут быть настроены, как показано ниже.

- * `elasticsearch` — используется для отправки событий напрямую в Elasticsearch.

Образец конфигурации вывода в Elasticsearch выглядит следующим образом:

```
output.elasticsearch:  
  enabled: true  
  hosts: ["localhost:9200"]
```

С помощью настройки `enabled` можно включать или выключать вывод. Параметр `hosts` принимает один или несколько узлов/серверов Elasticsearch. Множественные хосты могут быть указаны для достижения устойчивости при сбоях. В случаях, когда настроено несколько хостов, события доставляются по этим узлам в циклическом порядке. Если настроена авторизация Elasticsearch, данные аутентификации могут быть введены под параметрами `username` и `password`:

```
output.elasticsearch:  
  enabled: true  
  hosts: ["localhost:9200"]  
  username: "elasticuser"  
  password: "password"
```

Для доставки событий в контейнер узла поглощения Elasticsearch, чтобы их можно было предварительно обработать до хранения в Elasticsearch, информацию о контейнере указывают в параметре `pipeline`:

```
output.elasticsearch:  
  enabled: true  
  hosts: ["localhost:9200"]  
  pipeline: "nginx_log_pipeline"
```

Индекс по умолчанию, в который записываются данные, имеет формат `metricbeat-%{[beat.version]}-%{+yyyy.MM.dd}`. Таким образом, каждый день будет создаваться новый индекс. Например, сегодня 2 декабря 2017 года, и все события будут размещены в индексе `metricbeat-6.0.0-2017-12-02`. Вы можете изменить название индекса или шаблон, используя параметр `index`. В следующем фрагменте конфигурации новый индекс создается каждый месяц:

```
output.elasticsearch:  
  hosts: ["http://localhost:9200"]  
  index: "metricbeat-%{[beat.version]}-%  
  {+yyyy.MM}"
```

Используя параметр `indices`, вы можете распределять события по индексам в зависимости от выбранных условий. В следующем фрагменте кода указано, что, если сообщение содержит строку DEBUG, оно будет помещено в индекс `debug-%{+yyyy.MM.dd}`. Если сообщение содержит строку ERR, оно будет помещено в индекс `error-%{+yyyy.MM.dd}`. Если сообщение не содержит таких строк, оно будет помещено в индекс `logs-%{+yyyy.MM.dd}`, как определено в параметре `index`:

```
output.elasticsearch:  
  hosts: ["http://localhost:9200"]  
  index: "logs-%{+yyyy.MM.dd}"  
  indices:  
    - index: "debug-%{+yyyy.MM.dd}"  
      when.contains:  
        message: "DEBUG"  
    - index: "error-%{+yyyy.MM.dd}"  
      when.contains:  
        message: "ERR"
```



После того как параметр `index` изменен, отключите шаблоны и панели управления, добавив следующие настройки:

```
setup.dashboards.enabled: false  
setup.template.enabled: false
```

В качестве альтернативы вы можете указать следующие параметры в файле metricbeat.yml: `setup.template.name` и `setup.template.pattern`. В противном случае Metricbeat не запустится.

- `logstash` – используется для отправки событий в Logstash.



Для использования Logstash как источника вывода необходимо настроить его с плагином ввода Beats, чтобы получать входящие события Beats.

Пример конфигурации вывода в Logstash выглядит следующим образом:

```
output.logstash:  
  enabled: true  
  hosts: ["localhost:5044"]
```

С помощью настройки `enabled` возможно включать или выключать вывод. Параметр `hosts` принимает один или несколько узлов/серверов Logstash. Множественные хосты можно указывать для достижения устойчивости при сбоях. Если настроенный хост не отвечает, тогда события будут отправлены в один из других перечисленных в настройках хостов. В случаях, когда настроено несколько хостов, события доставляются по этим узлам в случайном порядке. Для включения балансировки нагрузки событий на хосты Logstash установите флаг `loadbalance` на значение `true`:

```
output.logstash:
```

```
        hosts:      ["localhost:5045",  
"localhost:5046"]  
        loadbalance: true
```

- `console` — применяется для отправки событий в `stdout`. События будут записаны в формате JSON. Предназначено для тестирования или отладки.

Пример конфигурации консоли выглядит следующим образом:

```
output.console:  
    enabled: true  
    pretty: true
```

Логирование

Этот раздел содержит настройки вывода логирования Metricbeat. Система может записывать логи в `syslog` или циклические лог-файлы. Если не было задано четких настроек логирования, в системах Windows будет использоваться вывод в файл, а в системах Linux и OS X — вывод `syslog`.

Пример конфигурации выглядит следующим образом:

```
logging.level: debug  
logging.to_files: true  
logging.files:  
    path: C:\logs\metricbeat  
    name: metricbeat.log  
    keepfiles: 10
```

Некоторые из настроек конфигурации:

- `level` — для указания уровня логирования;

- `to_files` – для записи всех логов в файл. Файлы подлежат ротации. Это значение по умолчанию;
- `to_syslog` – для записи логов в `syslog` значение должно быть равным `true`;
- `files.path`, `files.name` и `files.keepfiles` – используются для указания местонахождения файлов, их названий и количества недавно прошедших ротацию лог-файлов для хранения на диске.

Сбор системных метрик

Для мониторинга и сбора серверных метрик в Metricbeat предусмотрен модуль `system`. Он предоставляет следующие метрик-сеты для сбора серверных метрик.

- `core` – статистика использования для каждого ядра ЦП.
- `cpu` – статистика ЦП.
- `diskio` – I/O-метрики диска из операционной системы. Для каждого диска, монтированного в системе, создается одно событие.
- `filesystem` – статистика файловой системы. Для каждой файловой системы создается одно событие.
- `process` – статистика процессов. По событию на каждый процесс.
- `process_summary` – высокоуровневая статистика по выполняемым процессам.

- `fsstat` — общая статистика файловой системы.
- `load` — статистика загрузки.
- `memory` — статистика памяти.
- `network` — I/O-метрики сети из операционной системы. По событию для каждого сетевого интерфейса.
- `socket` — записывает событие для каждого нового TCP-сокета. Доступен только на платформе Linux, для его работы необходим kernel версии 2.6.14 или новее.

Некоторые из метрик-сетов предоставляют возможность настраивать возвращаемые метрики. Например, метрик-сет `sri` предоставляет конфигурацию `sri.metrics` для контроля метрик, полученных от ЦП. Однако такие метрик-сеты, как `memory` и `diskio`, не предоставляют никаких возможностей настройки. В отличие от остальных модулей, которые можно мониторить с других серверов путем соответствующей настройки хостов (не самый лучший вариант), модули `system` являются локальными для сервера и могут собирать метрики только в соответствующих хостах.



Полный список полей по метрик-сетам, которые экспортятся модулем `system`, можно найти по следующей ссылке:

<https://www.elastic.co/guide/en/beats/metricbeat/current/exported-fields-system.html>.

Запуск Metricbeat с модулем system

Используем Metricbeat для сбора системных метрик. Убедитесь, что запущены Kibana 6.0 и Elasticsearch 6.0.

1. Замените содержимое файла metricbeat.yml следующей конфигурацией и сохраните файл:

```
##### Metricbeat Configuration
Example #####
=====
Modules      configuration
=====

metricbeat.config.modules:
  # Glob pattern for configuration loading
  path: ${path.config}/modules.d/*.yml

  # Set to true to enable config reloading
  reload.enabled: false

  # Period on which files under path should
  # be checked for changes
  #reload.period: 10s

=====
Elasticsearch template setting
=====

setup.template.settings:
  index.number_of_shards: 1
  index.codec: best_compression
  #_source.enabled: false
```

```
#===== General
Settings=====
name: metricbeat_inst1

tags: ["system-metrics", "localhost"]

fields:
  env: test-env

#===== Dashboards
=====
setup.dashboards.enabled: true

#===== Kibana
Settings
=====
setup.kibana:
  host: "localhost:5601"
  #username: "elastic"
  #password: "changeme"
#----- Elasticsearch
output Settings -----
-----
output.elasticsearch:
  # Array of hosts to connect to.
  hosts: ["localhost:9200"]
  #username: "elastic"
  #password: "changeme"
```



Параметр `setup.dashboards.enabled: true` загрузит образцы панелей управления в индекс Kibana во время запуска; для загрузки используются Kibana API. Если у вас настроена безопасность для Elasticsearch и Kibana, убедитесь, что со строк `username` и `password` снят комментарий и заданы необходимые значения.

2. По умолчанию модуль `system` активизирован. Убедитесь в этом, выполнив следующую команду:

Windows :

```
D:\packt\metricbeat-6.0.0-windows-x86_64>metricbeat.exe modules  
enable system
```

Module system is already enabled

Linux :

```
[locationOfMetricBeat]$ ./metricbeat modules  
enable system
```

Module system is already enabled

3. Вы можете проверить активизированные метрик-сеты для модуля `system`, открыв файл `system.yml`, который находится в папке `modules.d`:

```
#system.yml  
- module: system  
  period: 10s  
  metricsets:  
    - cput
```

```

#- load
  - memory
  - network
  - process
  - process_summary
#- core
#- diskio
#- socket
processes: ['./*']
process.include_top_n:
  by_cpu: 5 # include top 5 processes by
CPU
  by_memory: 5 # include top 5 processes by
memory

- module: system
  period: 1m
metricsets:
  - filesystem
  - fsstat
processors:
  - drop_event.when.regexp:
    system.filesystem.mount_point:
      '^/(sys|cgroup|proc|dev|etc|host|lib)(/$|/)'

```

Как видно из предыдущего кода, конфигурация модуля указана дважды с разными периодами использования набора метрик-сетов.

Активизированы следующие метрик-сеты: `cput`, `memory`, `network`, `process`, `process_summary`, `filesystem`, `fsstats`.

4. Запустите Metricbeat, выполнив такую команду:

Windows :

```
D:\packt\metricbeat-6.0.0-windows-x86_64>metricbeat.exe -e
```

Linux :

```
[locationOfMetricBeat]$ ./metricbeat -e
```

Как только Metricbeat запущен, он загружает образцы панелей управления Kibana и начинает отправлять метрики в Elasticsearch. Для проверки запустите следующую команду:

```
curl -X GET 'http://localhost:9200/_cat/indices?v=&format=json'
```

Пример ответа:

```
[  
 {  
   "health": "yellow",  
   "status": "open",  
   "index": "metricbeat-6.0.0-2017.11.26",  
   "uuid": "w2WoP2IhQ9eG7vSU_HmgmA",  
   "pri": "1",  
   "rep": "1",  
   "docs.count": "29",  
   "docs.deleted": "0",  
   "store.size": "45.3kb",  
   "pri.store.size": "45.3kb"  
 },  
 {
```

```
        "health": "yellow",
        "status": "open",
        "index": ".kibana",
        "uuid": "sSzeYu-YTtWR8vr2nzKrbg",
        "pri": "1",
        "rep": "1",
        "docs.count": "108",
        "docs.deleted": "59",
        "store.size": "289.3kb",
        "pri.store.size": "289.3kb"
    }
]
```

```
curl -X GET 'http://localhost:9200/_cat/indices?v'
```

```
health status index uuid pri rep docs.count
docs.deleted store.size pri.store.size
yellow open metricbeat-6.0.0-2017.11.26
w2WoP2IhQ9eG7vSU_HmgmA 1 1 29 0 45.3kb 45.3kb
yellow open .kibana sSzeYu-YTtWR8vr2nzKrbg 1 1
108 59 289.3kb 289.3kb
```

Указание псевдонимов

Elasticsearch позволяет пользователям создавать *псевдонимы* — виртуальное название индекса, которое можно применять для ссылки на один или несколько индексов. API псевдонимов индексов Elasticsearch позволяет предоставить индексу псевдоним, который будет автоматически конвертироваться в реальное название индекса всеми API.

Например, мы хотим выполнить запрос для набора схожих индексов. Вместо того чтобы указывать в запросе название каждого индекса, мы можем использовать псевдонимы. Псевдоним будет указывать на все индексы, и вы сможете выполнить запрос по ним. Это очень удобно: мы станем добавлять определенные индексы динамически на регулярной основе таким образом, чтобы приложение или пользователь, выполняющие запрос, не волновались о включении этих индексов в запрос. Для этого потребуется обновить индекс псевдонимом (указывает вручную администратор или задается во время создания индекса).

Предположим, ИТ-администратор создал псевдоним, который ссылается на все индексы, содержащие метрики по определенному месяцу. Например, как указано в следующем фрагменте кода, псевдоним с названием `november_17_metrics` создан для всех индексов шаблона `metricbeat-6.0.0-2017.11.*`, то есть индексов Metricbeat за каждый день ноября:

```
curl -X POST http://localhost:9200/_aliases -H
'content-type:
application/json' -d '{
  "actions": [
    {
      "add": {
        "index" : "metricbeat-6.0.0-
2017.11.*",
        "alias": "november_17_metrics"
      }
    }
}'
```

Теперь, используя псевдоним `november_17_metrics`, можно выполнить запрос по всем индексам шаблона `metricbeat-6.0.0-2017.11.*`:

```
curl -X GET http://localhost:9200/november_17_metrics/_search
```

В следующем примере псевдоним `sales` создан для индексов `it_sales` и `retail_sales`. В будущем при создании нового индекса продаж он также будет относиться к индексу `sales`. Следовательно, конечный пользователь/приложение может указывать конечную точку `sales` для запросов по всем данным о продажах:

```
curl -X POST http://localhost:9200/_aliases -d
'{
  "actions" : [
    { "add" : { "index" : "it_sales", "alias" : "sales" } },
    { "add" : { "index" : "retail_sales", "alias" : "sales" } }
  ]
}'
```

Для удаления псевдонима индекса используйте действие `remove` в API псевдонимов:

```
curl -X POST http://localhost:9200/_aliases -d
'{
  "actions" : [ { "remove" : { "index" : "retail_sales", "alias" : "sales" } } ]
}'
```

Визуализация системных метрик в Kibana

Для визуализации системных метрик в Kibana выполните следующие шаги.

- Перейдите по ссылке <http://localhost:5601> и откройте Kibana.
- Щелкните на ссылке **Dashboard** (Панель управления), которая находится в левом навигационном меню, и выберите **[Metricbeat System] Overview** или **[Metricbeat System] Host Overview** (рис. 11.3).

Рис. 11.3

[Metricbeat System] Overview. Эта панель управления предоставляет обзор всех систем, находящихся под мониторингом. Поскольку у нас в распоряжении только один хост, мы увидим количество хостов, равное 1 (рис. 11.4).

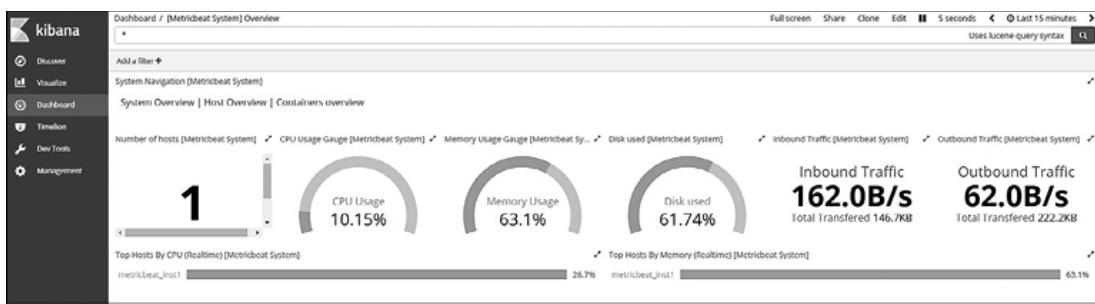


Рис. 11.4

[Metricbeat System] Host Overview. Эта панель управления полезна для поиска подробных метрик по выбранным

системам/хостам. Для фильтрации метрик по определенному хосту добавьте соответствующий критерий поиска/фильтрации на панели поиска/запроса. На рис. 11.5 критерий поиска — `beat.name:metricbeat_inst1`. Вы можете использовать любой атрибут, который позволяет уникально идентифицировать систему/хост. Например, можно добавить фильтр на основе `beat.hostname`.

Поскольку метрик-сеты `diskio` и `load` были отключены в конфигурации системного модуля, мы увидим пустые визуализации для системной нагрузки и I/O-данных по диску (рис. 11.6).

Для просмотра панели управления в реальном времени найдите настройки в правом верхнем углу и выберите необходимый интервал обновления (рис. 11.7).

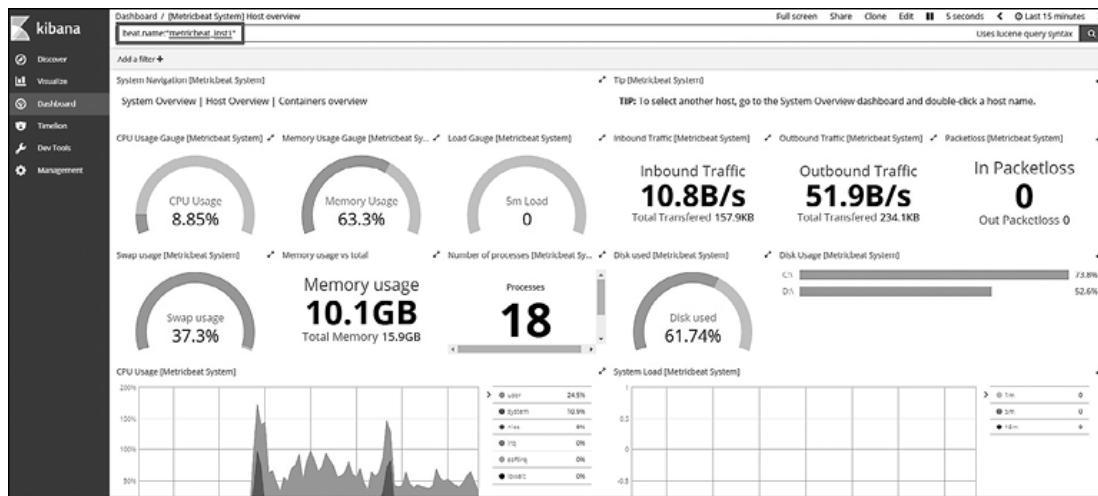


Рис. 11.5

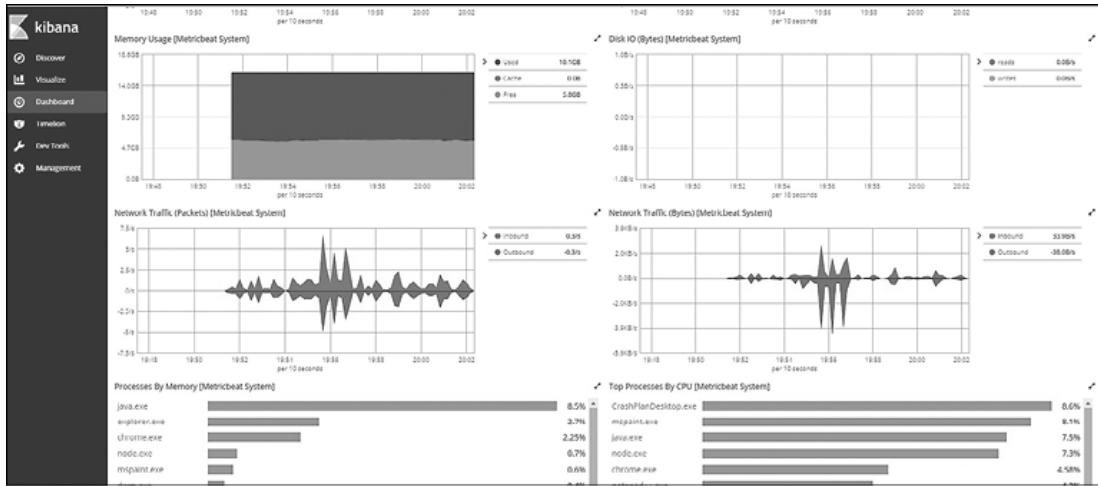


Рис. 11.6



Рис. 11.7



Для просмотра панели управления в полноэкранном режиме нажмите кнопку Full screen (Полноэкранный режим) вверху навигационной панели. В результате браузер и верхняя навигационная панель будут спрятаны. Для возврата к обычному режиму нажмите кнопку Kibana снизу слева или просто клавишу Esc.



Более подробно об эффективном использовании Kibana для получения развернутой информации о ваших данных рассказывается в главе 7.

Архитектура внедрения

На рис. 11.8 изображена распространенная архитектура внедрения Elastic Stack.

Существует три возможных архитектуры.

- **Отправка операционных метрик напрямую в Elasticsearch.** Как видно на схеме, на пограничных серверах, с которых необходимо доставлять операционные метрики/логи, будут установлены различные компоненты Beats, такие как Metricbeat, Filebeat, Packetbeat и пр. Если не требуется дальнейшая обработка событий, тогда полученные события будут отправлены напрямую в кластер Elasticsearch. Как только данные появляются в Elasticsearch, их можно визуализировать или анализировать с помощью Kibana. В этой архитектуре поток событий будет следующим: Beats—>Elasticsearch—>Kibana.
- **Отправка операционных метрик в Logstash.** Собранные компонентами Beats (установленными на пограничных серверах) операционные метрики/логи отправляются в Logstash для дальнейшей обработки. После этого обработанные/дополненные события отправляются в Elasticsearch. Для увеличения объема обработки можно масштабировать экземпляры Logstash, например настроить

одну часть компонентов Beats так, чтобы отправлять данные в экземпляр Logstash 1, другую часть компонентов — отправлять данные в экземпляр Logstash 2 и т.д. В этой архитектуре поток событий будет следующим: Beats—>Logstash—>Elasticsearch—>Kibana.

- **Отправка операционных метрик в отказоустойчивую очередь.** Если события создаются на высокой скорости и Logstash не справляется с нагрузкой, вы можете направить поток данных в отказоустойчивые очереди, такие как Apache Kafka, для предотвращения потерь данных. События встанут в очередь, после чего Logstash сможет обработать их на своей скорости, тем самым можно избежать потери операционных метрик/логов, собранных компонентами Beats. В этой архитектуре поток событий будет следующим: Beats—>Kafka—>Logstash—>Elasticsearch—>Kibana.



В версии Logstash 5.x можно настроить отказоустойчивую очередь средствами Logstash. Тем не менее она не имеет такого высокого уровня устойчивости, как Kafka.

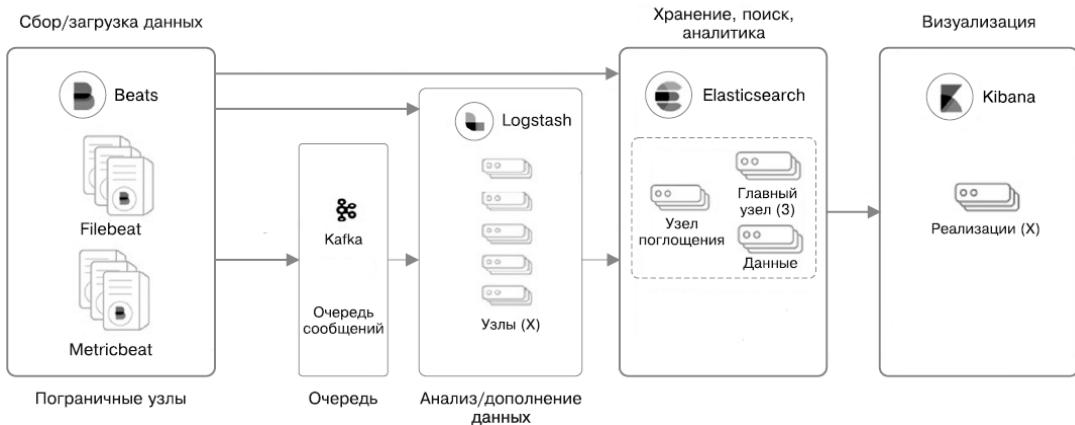


Рис. 11.8

В указанных выше архитектурах можно выполнять простое масштабирование вверх или вниз для реализаций Elasticsearch, Logstash и Kibana, в зависимости от сферы применения.

Резюме

В этой главе мы подробно рассмотрели еще один компонент Beats под названием Metricbeat. Мы разобрались, как установить и настроить Metricbeat так, чтобы он мог отправлять операционные метрики в Elasticsearch. Мы также рассмотрели различные архитектуры внедрения для создания решений мониторинга в реальном времени на базе Elastic Stack, которые могут быть полезны для мониторинга серверов и приложений. Это поможет ИТ-администраторам и сотрудникам поддержки приложений получать подробную информацию о поведении приложений и серверов и позволит им оперативно реагировать в случае выхода из строя инфраструктуры.