POLITECHNIKA GDAŃSKA | WYDZIAŁ ELEKTRONIKI, TELEKOMUNIKACJI I INFORMATYKI

Dokumentacja Prezentacji

Elasticsearch

Wydział Elektroniki, Telekomunikacji i Informatyki

Politechnika Gdańska

Technologia:	
Elasticsearch search engine	
Zespół projektowy:	**Promotor prezentacji:**
Damian Strojek, s184407	*dr inż. Jerzy Demkowicz*

Krótki opis technologii:

Elasticsearch to wyszukiwarka oparta na bibliotece Lucene. Zapewnia rozproszoną, wielodostępną wyszukiwarkę pełnotekstową z interfejsem WWW HTTP i dokumentami JSON bez schematu. Elasticsearch jest również platformą wyszukiwania w czasie zbliżonym do rzeczywistego, co oznacza, że opóźnienie od momentu indeksowania dokumentu do momentu jego wyszukiwania jest bardzo krótkie.

Spis treści

1 O technologii

W dzisiejszych czasach niezwykle ważne jest szybkie docieranie do interesujących nas informacji. Niezależnie od tematu każdy chce mieć odpowiedź na wyciągnięcie ręki. Wyszukiwanie ma szczególne znaczenie w, np. środowisku internetowym oraz w samym procesie tworzenia narzędzi informatycznych.

Aplikacje bardzo często posiadaą ogromną ilość zapisanych danych. Oczywiście, aby znaleźć te odpowiednie, trzeba je przeszukiwać. Istotna jest więc kolejność wykonywanych kroków oraz sama struktura zapisu. Szczególnie ważne jest to w przypadku gdy mamy do czynienia z różnymi rodzajami danych – nie każda technologia jest przystosowana do radzenia sobie z różniącymi się od siebie formatami. W takim przypadku możemy spojrzeć w stronę produktu od developerów Elastic NV.

Elasticsearch (Elasticsearch Github Repo) to baza danych typu NoSQL (nierelatywna) służąca wyszukiwaniu informacji. Jest ona silnikiem wyszukiwania „pełnotekstowego” i wykorzystuje bibliotekę Apache Lucene (biblioteka oprogramowania wyszukiwarek typu open source, napisana w Javie - tak samo jak Elasticsearch). Dzięki takiemu połączeniu jest to potężne narzędzie pozwalające na kompleksowe przeszukiwanie danych w czasie rzeczywistym. Technologia ta jest podwójnie licencjonowana na podstawie source-available Server Side Public License oraz Elastic license.

2 Historia

Shay Banon stworzył prekursora Elasticsearch, zwanego Compass, w 2004 roku. Myśląc o trzeciej wersji Compass, zdał sobie sprawę, że konieczne będzie przepisanie dużych części Compassa, aby "stworzyć skalowalne rozwiązanie wyszukiwania". Stworzył więc "rozwiązanie zbudowane od podstaw do dystrybucji" i użył wspólnego interfejsu, JSON over HTTP, odpowiedniego również dla języków programowania innych niż Java. Shay Banon wydał pierwszą wersję Elasticsearch w lutym 2010 roku.

Firma Elastic NV została założona w 2012 roku w celu świadczenia usług komercyjnych i dostarczania produktów związanych z Elasticsearch i powiązanym oprogramowaniem.

3 Charakterystyka Elasticsearch

Elasticsearch może być używany do wyszukiwania dowolnego rodzaju dokumentu. Zapewnia skalowalne wyszukiwanie, w czasie zbliżonym do rzeczywistego i obsługuje wielodostępność. Dodatkowo, można do każdego pola dołożyć wagę i tym samym wpłynąć na scoring zapytania.

4 Architektura Elasticsearch

Elasticsearch jest rozproszony, co oznacza, że indeksy można podzielić na odłamki, a każdy fragment może mieć zero lub więcej replik. Każdy węzeł obsługuje jeden lub więcej fragmentów i działa jako koordynator delegujący operacje do właściwego fragmentu (fragmentów). Równoważenie i routing odbywają się automatycznie. Powiązane dane są często przechowywane w tym samym indeksie, który składa się z jednego lub więcej odłamków pierwotnych oraz zerowych lub więcej fragmentów repliki. Po utworzeniu indeksu nie można zmienić liczby fragmentów podstawowych.

Elasticsearch jest rozwijany wraz z silnikiem gromadzenia danych i analizowania dzienników Logstash, platformą analityczną i wizualizacyjną Kibana oraz kolekcją lekkich spedytorów danych o nazwie Beats. Cztery produkty są przeznaczone do użytku jako zintegrowane rozwiązanie, określane jako "Elastic Stack". (Dawniej "ELK stack", skrót od "Elasticsearch, Logstash, Kibana".)

Elasticsearch używa Lucene i stara się udostępnić wszystkie jego funkcje za pośrednictwem JSON i Java API. Obsługuje facetting (rozszerzenie wyszukiwania leksykalnego o fasetowy system nawigacji, umożliwiając użytkownikom zawężanie wyników poprzez stosowanie filtrów) i perkolację (formę wyszukiwania prospektywnego), co może być przydatne do powiadamiania, jeśli nowe dokumenty pasują do zarejestrowanych zapytań. Inna funkcja, "brama", obsługuje długoterminową trwałość indeksu; Na przykład indeks można odzyskać z bramy w przypadku awarii serwera. Elasticsearch obsługuje żądania GET w czasie rzeczywistym, co czyni go odpowiednim magazynem danych NoSQL, ale brakuje mu rozproszonych transakcji.

4.1 Kluczowe koncepty

Dogłębne zrozumienie architektury elasticsearcha wymaga zapoznania się z kluczowymi konceptami, które razem tworzą jedno, zsynchronizowane środowisko.

Do konceptów tych należą:

- Node (węzeł) – odnosi się do pojedynczej uruchomionej instancji Elasticsearch. Pojedynczy serwer fizyczny lub wirtualny może pomieścić wiele węzłów w zależności od możliwości ich zasobów fizycznych, takich jak pamięć RAM, pamięć masowa i moc obliczeniowa.
- Cluster (klaster) – zbiór jednego lub więcej węzłów. Klaster zapewnia zbiorcze możliwości indeksowania i wyszukiwania we wszystkich węzłach dla całych danych.
- Index (indeks) – zbiór różnego rodzaju dokumentów i ich właściwości. Index wykorzystuje również koncepcję odłamków (shardów) w celu poprawy wydajności.
- Document (dokument) – zbiór pól w określony sposób zdefiniowany w formacie JSON. Każdy dokument należy do typu i znajduje się wewnątrz indeksu. Każdy dokument jest również powiązany z unikatowym identyfikatorem zwanym identyfikatorem UID.
- Shard (odłamek) – indeksy są podzielone na odłamki. Oznacza to, że każdy fragment zawiera wszystkie właściwości dokumentu, ale zawiera mniejszą liczbę obiektów niż indeks. Separacja pozioma sprawia, że odłamek jest niezależnym węzłem, który można przechowywać w dowolnym węźle. Pierwotny fragment jest oryginalną poziomą częścią indeksu, a następnie te pierwotne odłamkki są replikowane na odłamki repliki. Więcej na ten temat w sekcji *4.2 Odłamki*.
- Replicas (repliki) – Elasticsearch pozwala użytkownikowi tworzyć repliki swoich indeksów i fragmentów. Replikacja nie tylko pomaga w zwiększeniu dostępności danych w przypadku awarii, ale także poprawia wydajność wyszukiwania poprzez przeprowadzene operacji wyszukiwania równoległego w tych replikach.
- Inverted index (indeks odwrócony) – indeks bazy danych przechowujący mapowanie z treści, takiej jak slowa lub liczby, do jej lokalizacji w tabeli lub w dokumencie. Celem indeksu odwróconego jest umożliwienie szybkiego wyszukiwania pełnotekstowego, kosztem zwiększonego przetwarzania, gdy dokument jest dodawany do bazy danych.

4.2 Koncept odłamków

Odłamki (często później nazywane „shardami”), mogą być rozumiane jako **instancje indeksów Apache Lucene**, które można traktować jako samodzielną wyszukiwarkę, która indeksuje i obsługuje zapytania dotyczące podzbioru danych w klastrze Elasticsearch. Każdy indeks składa się z jednego lub więcej odłamków. Odłamek może być po prostu odłamkiem, odłamkiem pierwotnym lub repliką.

Gdy dane są zapisywane w odłamkach, są one okresowo publikowane w nowych, niezmiennych segmentach Lucene na dysku i w tym momencie stają się dostępne do zapytania. Jest to określane jako odświeżenie. Jak to działa, opisano bardziej szczegółowo w elastic/inside-a-shard.

Wraz ze wzrostem liczby segmentów są one okresowo konsolidowane w większe segmenty. Proces ten nazywany jest merge’owaniem. Ponieważ wszystkie segmenty są niezmienne, oznacza to, że używane miejsce na dysku zwykle będzie się zmieniać podczas indeksowania, ponieważ nowe, połączone segmenty muszą zostać utworzone, zanim te, które zastępują, będą mogły zostać usunięte. Scalanie może wymagać znacznych zasobów, zwłaszcza w odniesieniu do dysku we/wy.

Odłamek można również określić jako jednostka, w której Elasticsearch dystrybuuje dane w klastrze. Szybkość, z jaką Elasticsearch może przenosić odłamki podczas równoważenia danych, np. po awarii, zależy od rozmiaru i liczby fragmentów, a także wydajności sieci i dysku.

Podczas tworzenia indeksu można określić liczbę odłamków i liczbę replik na fragment. Wartość domyślna to 5 odłamków podstawowych i 1 replika na odłamek. Odłamki są automatycznie równomiernie rozmieszczone w klastrze. Odłamek repliki nigdy nie zostanie przydzielony na tym samym komputerze, na którym znajduje się powiązany odłamek główny.

To, ile fragmentów i replik zostanie użytych, zależy tak naprawdę od danych, sposobu dostępu do nich i liczby dostępnych węzłów / serwerów. Najlepszym rozwiązaniem jest nieznaczne nadmierne przydzielanie fragmentów w celu ich redystrybucji w przypadku dodania większej liczby węzłów do klastra, ponieważ nie można (na razie) zmienić liczby fragmentów po utworzeniu indeksu.

W przeciwnym razie zawsze można zmienić liczbę fragmentów, jeśli chce się wykonać całkowite ponowne indeksowanie danych.

Każdy dodatkowy odłamek wiąże się z kosztami, ponieważ każdy odłamek jest w rzeczywistości instancją Lucene. Maksymalna liczba odłamków, które możesz mieć na maszynę, zależy od dostępnego sprzętu i danych. Dobrze wiedzieć, że posiadanie 100 indeksów z każdym odłamkiem lub jednego indeksu ze 100 odłamkami jest naprawdę takie samo, ponieważ w obu przypadkach istnieje 100 instancji lucene.

Oczywiście w czasie zapytania, jeśli chcemy wysłać zapytanie do pojedynczego indeksu elasticsearch złożonego ze 100 fragmentów, Elasticsearch musiałby wysłać zapytanie do nich wszystkich, aby uzyskać prawidłowe wyniki (chyba, że użyto określonego routingu dla dokumentów, aby następnie zapytać tylko o określony fragment). Wiązałoby się to z kosztami wydajności.

4.2.1 Indeks według okresu przechowywania

Ponieważ segmenty są niezmienne, aktualizacja dokumentu wymaga, aby Elasticsearch najpierw znalazł istniejący dokument, a następnie oznaczył go jako usunięty i dodał zaktualizowaną wersję. Usunięcie dokumentu wymaga również znalezienia i oznaczenia go jako usuniętego. Z tego powodu usunięte dokumenty będą nadal wiązać miejsce na dysku i niektóre zasoby systemowe, dopóki nie zostaną scalone, co może zużywać dużo zasobów systemowych.

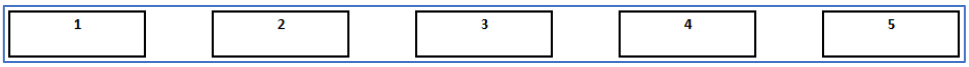
Elasticsearch umożliwia bardzo wydajne usuwanie pełnych indeksów bezpośrednio z systemu plików, bez konieczności jawnego usuwania wszystkich rekordów indywidualnie. Jest to zdecydowanie najskuteczniejszy sposób usuwania danych z Elasticsearch.

Dobra praktyka to używanie indeksów opartych na czasie do zarządzania przechowywaniem danych, gdy tylko jest to możliwe. Grupuj dane w indeksy na podstawie okresu przechowywania. Indeksy oparte na czasie ułatwiają również zmianę liczby podstawowych odłamków i replik w czasie, ponieważ można to zmienić w celu wygenerowania następnego indeksu. Upraszcza to dostosowywanie się do zmieniających się ilości danych i wymagań.

4.2.2 Przykład utworzenia odłamka

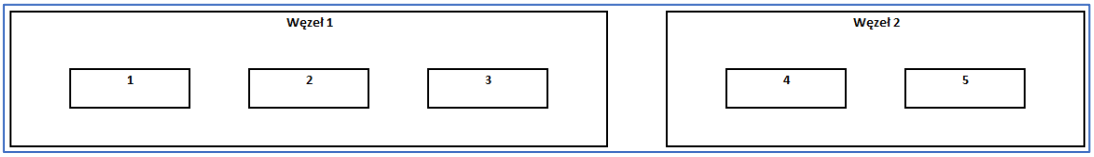
Załóżmy, że utworzono nowy klaster z jednym węzłem, który właśnie został uruchomiony. Nie mamy danych, dlatego musimy stworzyć indeks.

Podczas tworzenia indeksu (indeks jest tworzony automatycznie podczas indeksowania również pierwszego dokumentu) można określić, z ilu fragmentów będzie się składał. Jeśli liczba nie zostanie określona, będzie ona miała domyślną liczbę odłamków: 5 podstawowych. Oznacza to, że elasticsearch utworzy 5 podstawowych fragmentów, które będą zawierać dane:



Za każdym razem, gdy dokument jest poddawany indeksowaniu, elasticsearch zdecyduje, który fragment podstawowy ma pomieścić ten dokument i zindeksuje go tam. Odłamki pierwotne nie są kopią danych, są danymi. Posiadanie wielu fragmentów pomaga w wykorzystaniu przetwarzania równoległego na jednej maszynie, ale chodzi o to, że jeśli uruchomimy kolejną instancję Elasticsearch na tym samym klastrze, fragmenty zostaną równomiernie rozłożone w klastrze.

Węzeł 1 będzie wtedy zawierał na przykład tylko trzy odłamki, ponieważ pozostałe dwa odłamki zostały przeniesione do nowo uruchomionego węzła.

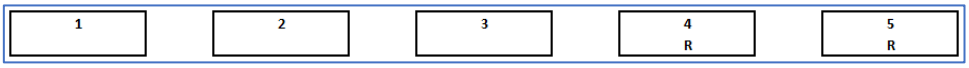


Dlaczego tak się dzieje? Ponieważ elasticsearch jest rozproszoną wyszukiwarką i w ten sposób można korzystać z wielu węzłów / maszyn do zarządzania dużymi ilościami danych.

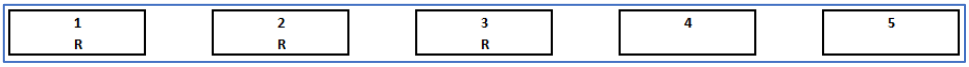
Każdy indeks elasticsearch składa się z co najmniej jednego fragmentu podstawowego, ponieważ to tam przechowywane są dane. Każdy odłamek ma jednak swoją cenę, dlatego jeśli posiadamy pojedynczy węzeł i nie mamy przewidywalnego wzrostu, jeden podstawowy odłamek powinien wystarczyć.

Innym rodzajem odłamka jest replika. Wartość domyślna to 1, co oznacza, że każdy fragment podstawowy zostanie skopiowany do innego fragmentu, który będzie zawierał te same dane. Repliki służą do zwiększania wydajności wyszukiwania i przełączania awaryjnego. Fragment repliki nigdy nie zostanie przydzielony w tym samym węźle, w którym znajduje się powiązany element podstawowy (byłoby to prawie jak umieszczenie kopii zapasowej na tym samym dysku, co oryginalne dane).

Wracając do naszego przykładu, przy 1 replice będziemy mieli cały indeks na każdym węźle, ponieważ 2 odłamki repliki zostaną przydzielone na pierwszym węźle i będą zawierać dokładnie te same dane, co podstawowe odłamki w drugim węźle:



To samo dotyczy drugiego węzła, który będzie zawierał kopię podstawowych odłamków w pierwszym węźle:



W przypadku takiej konfiguracji, jeśli węzeł ulegnie awarii, nadal posiadamy cały indeks. Odłamki repliki automatycznie staną się podstawowe, a klaster będzie działał poprawnie pomimo awarii węzła.

4.3 Koncept indeksu odwróconego

W informatyce indeks odwrócony (określany również jako lista ogłoszeń, plik ogłoszeń lub plik odwrócony) jest indeksem bazy danych przechowującym mapowanie z zawartości, takiej jak słowa lub liczby, do jej lokalizacji w tabeli lub w dokumencie lub zestawie dokumentów (nazwanych w przeciwieństwie do indeksu do przodu, który mapuje z dokumentów na zawartość).

Celem odwróconego indeksu jest umożliwienie szybkiego wyszukiwania pełnotekstowego, co wiąże się ze zwiększonym przetwarzaniem po dodaniu dokumentu do bazy danych. Odwrócony plik może być samym plikiem bazy danych, a nie jego indeksem. Jest to najpopularniejsza struktura danych wykorzystywana w systemach wyszukiwania dokumentów, wykorzystywana na dużą skalę, na przykład w wyszukiwarkach. Ponadto, kilka znaczących systemów zarządzania bazami danych, opartych na komputerach mainframe ogólnego przeznaczenia, korzystało z architektur odwróconej listy, w tym ADABAS, DATACOM/DB i Model 204.

Istnieją dwa główne warianty indeksów odwróconych: Odwrócony indeks na poziomie rekordu (lub odwrócony indeks pliku lub po prostu odwrócony plik) zawiera listę odwołań do dokumentów dla każdego słowa. Odwrócony indeks na poziomie wyrazu (lub pełny indeks odwrócony lub lista odwrócona) zawiera dodatkowo pozycje każdego słowa w dokumencie. Ta ostatnia forma oferuje większą funkcjonalność (jak wyszukiwanie fraz), ale wymaga większej mocy obliczeniowej i przestrzeni do stworzenia.

Ze względów historycznych kompresja odwróconej listy i kompresja bitmapowa zostały opracowane jako oddzielne linie badań, a dopiero później zostały uznane za rozwiązujące zasadniczo ten sam problem.

4.3.1 Zastosowania indeksu odwróconego

Struktura danych odwróconego indeksu jest centralnym składnikiem typowego algorytmu indeksowania wyszukiwarek. Celem implementacji wyszukiwarki jest optymalizacja szybkości zapytania: znajdź dokumenty, w których występuje słowo X. Po opracowaniu indeksu forward, który przechowuje listy słów w dokumencie, jest on następnie odwracany, aby opracować odwrócony indeks. Wykonywanie kwerend dotyczących indeksu do przodu wymagałoby sekwencyjnej iteracji każdego dokumentu i każdego słowa w celu zweryfikowania pasującego dokumentu. Czas, pamięć i zasoby przetwarzania do wykonania takiego zapytania nie zawsze są technicznie realistyczne. Zamiast wymieniać słowa na dokument w indeksie do przodu, opracowywana jest struktura danych odwróconego indeksu, która zawiera listę dokumentów na słowo.

Po utworzeniu odwróconego indeksu zapytanie można rozwiązać, przeskakując do identyfikatora słowa (poprzez losowy dostęp) w odwróconym indeksie.

W czasach przedkomputerowych konkordancje (alfabetyczny spis wyrazów, zdań, znaków lub symboli znajdujących się w jakimś dziele literackim lub zbiorze dokumentów czy ksiąg) do ważnych książek były składane ręcznie. Były to skutecznie odwrócone indeksy z niewielką ilością towarzyszących komentarzy, których stworzenie wymagało ogromnego wysiłku.

W bioinformatyce odwrócone indeksy są bardzo ważne w sekwencji krótkich fragmentów zsekwencjonowanego DNA. Jednym ze sposobów znalezienia źródła fragmentu jest poszukiwanie go w referencyjnej sekwencji DNA. Niewielką liczbę niezgodności (wynikających z różnic między zsekwencjonowanym DNA a referencyjnym DNA lub błędów) można wyjaśnić, dzieląc fragment na mniejsze fragmenty – co najmniej jeden podfragment prawdopodobnie pasuje do referencyjnej sekwencji DNA. Dopasowanie wymaga skonstruowania odwróconego indeksu wszystkich podciągów o określonej długości z referencyjnej sekwencji DNA. Ponieważ ludzkie DNA zawiera ponad 3 miliardy par zasad i musimy przechowywać podciąg DNA dla każdego indeksu i 32-bitową liczbę całkowitą dla samego indeksu, wymóg przechowywania takiego odwróconego indeksu prawdopodobnie wynosiłby dziesiątki gigabajtów.

4.3.2 Przykład indeksu odwróconego

Załóżmy, że mamy tabelę w naszej bazie danych. Oto jak będzie wyglądać tabela `Quotes`:

Quote_id	***Quote_text***
101	Nadchodzi zima
102	Chaos to drabina
103	Idziesz, mój panie?
104	Nadeszła zima

Indeks odwrócony dla tabeli `Quotes` będzie wyglądał w następujący sposób:

word	***Quote_id***
Zima	*101, 104*
To	*102*
Nadchodzi	*101*
Nadeszła	*104*
Chaos	*102*
Drabina	*102*
Idziesz	*103*
Mój	*103*
Panie	*103*

Po skonstruowaniu indeksu, jak pokazano w tej tabeli, możemy znaleźć wszystkie cytaty z terminem "zima" za pomocą szybkiego wyszukiwania.

4.3.3 Poprawa indeksu odwróconego

Chociaż podstawowy indeks odwrócony może odpowiadać na zapytania, które mają *dokładne* dopasowanie w bazie danych, może on nie działać we wszystkich scenariuszach. Na przykład:

- Użytkownicy mogą wyszukiwać termin, który nie występuje dokładnie w odwróconym indeksie, ale nadal jest z nim powiązany. Na przykład wyszukiwanie „śniegu” lub „śnieży” zamiast „opadów śniegu”. Możemy rozwiązać ten problem poprzez **Stemming**, który jest techniką, która wyodrębnia formę źródłową słów poprzez usunięcie afiksów. Na przykład rdzenna forma słów „jedzenie”, „zjadł” i „zjedzony” to jeść.
- Mogą też wyszukać synonim. Aby rozwiązać ten problem, synonimy wyszukiwanego terminu są również wyszukiwane w odwróconym indeksie.

- Użytkownicy zazwyczaj szukają fraz, a nie pojedynczych słów. Aby ułatwić wyszukiwanie fraz, odwrócone indeksy na poziomie programu Word rejestrują również pozycję słowa w dokumencie, aby poprawić wyniki wyszukiwania.

5 Przykłady użycia

5.1 University of Oxford

Oxford, który zajął pierwsze miejsce na świecie w światowych rankingach uniwersytetów Times Higher Education (THE) w latach 2017, 2018, 2019 i 2020, jest bardzo konkurencyjny. Około 21 500 osób ubiegało się o około 3300 miejsc na studiach licencjackich w 2018 r. Zespół Oxford Information Technology składa się z 800 członków w ponad 110 jednostkach i obsługuje jedną z największych sieci prywatnych w kraju z ~100 000 zarejestrowanych urządzeń i redundantnym łączem Internetowym o prędkości zbliżonej do 40 Gbit/s.

Zespół Oxfordzki zajmujący się dziedziną „Incident response" stworzył Proof of Concept przy użyciu Elasticsearch 1.5.1 Open Source w roku 2015 i postanowili aktualizować go przez kolejne lata. To co chcieli osiągnąć to rozwój wewnętrzny SIEM nowej generacji przy użyciu Elastic Stack i udało im się to w oszałamiającym tempie. System, który stworzyli, nosi nazwę SAVANT.

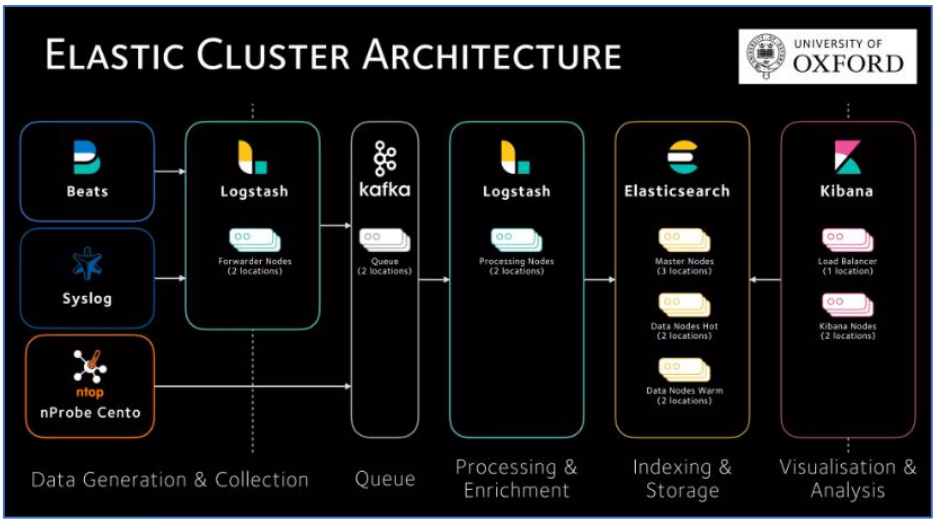
SAVANT ma na celu gromadzenie dzienników, audytów, ruchu, metryk i innych informacji w jednym centralnym miejscu do analizy bezpieczeństwa. System ten jest platformą analityczno-wizualizacyjną typu open source zaprojektowaną do współpracy z Elasticsearch. Może być używany do wyszukiwania, przeglądania i interakcji z danymi przechowywanymi w indeksach Elasticsearch. Może łatwo przeprowadzać zaawansowaną analizę danych i wizualizować dane na różnych wykresach, tabelach i mapach.

Jego prosty, oparty na przeglądarce interfejs umożliwia użytkownikowi przeglądanie dynamicznych pulpitów nawigacyjnych, które wyświetlają zmiany w zapytaniach Elasticsearch w czasie rzeczywistym. Pulpit nawigacyjny SAVANT wyświetla zbiór wizualizacji i wyszukiwań, które można dostosować do wymagań jednostki. Każdy publiczny interfejs został zaprojektowany z myślą o skalowalności, aby umożliwić rozwój bez wymuszania zmian na istniejących użytkownikach.

SAVANT jest prezentowany Uniwersytetowi w oparciu o dwa komponenty:

- Potok przetwarzania danych przy użyciu Logstash do pozyskiwania, przekształcania i wzbogacania danych z różnych źródeł, takich jak Beats, Syslog, NetStream;
- Analiza, wizualizacja i raportowanie danych przy użyciu Kibana i Elasticsearch API. Dostęp do danych jest zarządzany przez University SSO.

Poniżej znajduje się architektura, która pozwoliła Oxfordowi osiągnąć cel postawiony w 2015 roku.



Rys. 1 – Architektura Oxfordzkiej implementacji ES

5.2 U.S. Air Force

Siły Powietrzne U.S. dostrzegają potrzebę wytrwałych „Cyber Obrońców" zajmujących się zapewnieniem bezpieczeństwa systemom wykorzystywanym w misjach. W 2016 r. departament odpowiedział na to wezwanie, rozpoczynając prace poszukiwawcze dla Mission Defense Team (MDT). Jako lider dla Air Force Cyber, Air Combat Command (ACC) ma obowiązek organizować, szkolić i wyposażać siły cyberobrony w najskuteczniejsze i najbardziej odpowiednie narzędzia do wykonywania ich wyjątkowyh misji.

Elastic jest częścią zestawu narzędzi dostarczonego przez biuro ds. oceny luk w cyberprzestrzeni / Hunter Program Office for Defensive Cyber Operations, które ACC z powodzeniem wykorzystało w wielu misjach. Aktualna wizja to rozszerzenie wykorzystania Elastic na inne potencjalne wektory zagrożeń, w tym dane MIL-STD-1533, AIRINC-429 i SCADA.

Przykładem SCADA (Supervisory Control and Data Acquisition) mogą być Przemysłowe systemy sterowania (ICS). NIST definiuje ICS jako systemy informacyjne, które kontrolują procesy przemysłowe, takie jak produkcja, obsługa produktów, produkcja i dystrybucja. Składają się one z kombinacji elementów sterujących (np. elektrycznych, mechanicznych, hydraulicznych, pneumatycznych), które współpracują ze sobą, aby osiągnąć cele przemysłowe. Systemy te obsługują podstawowe sektory infrastruktury krytycznej, takie jak komunikacja, energetyka, transport, przemysł chemiczny i jądrowy.

Oprócz tego, że są podstawą krytycznej infrastruktury krajowej, co czyni je głównym celem cyberataków, systemy ICS mogą również wiązać się z wysokim fizycznym narażeniem ich zasobów i sieci, co sprawia, że ich monitorowanie i obrona bezpieczeństwa są znacznie szersze i bardziej złożone.

Ponadto zabezpieczenia ICS mają bardziej szczegółowe wymagania niż zabezpieczenia korporacyjne, co sprawia, że zastosowanie wspólnych mechanizmów bezpieczeństwa przedsiębiorstwa do ICS jest szczególnie negatywne dla funkcjonalności i bezpieczeństwa systemu.

Na przykład, triada bezpieczeństwa IT Confidentiality, Integrity, Availability (CIA) jest dla systemu ICS „do góry nogami" co sprawia, że cele programów bezpieczeństwa IT i ICS są bardzo różne. Operacyjny charakter środowisk wiąże się również z innymi wyzwaniami, takimi jak starsze i wycofane oprogramowanie oraz niemożność załatania lub uaktualnienia w celu natychmiastowego usunięcia luk w zabezpieczeniach. Może to narazić duże systemy na ryzyko przez miesiące, a nawet lata bez dostosowanych do potrzeb mechanizmów bezpieczeństwa, które łączą zarówno bezpieczeństwo, jak i funkcjonalność.

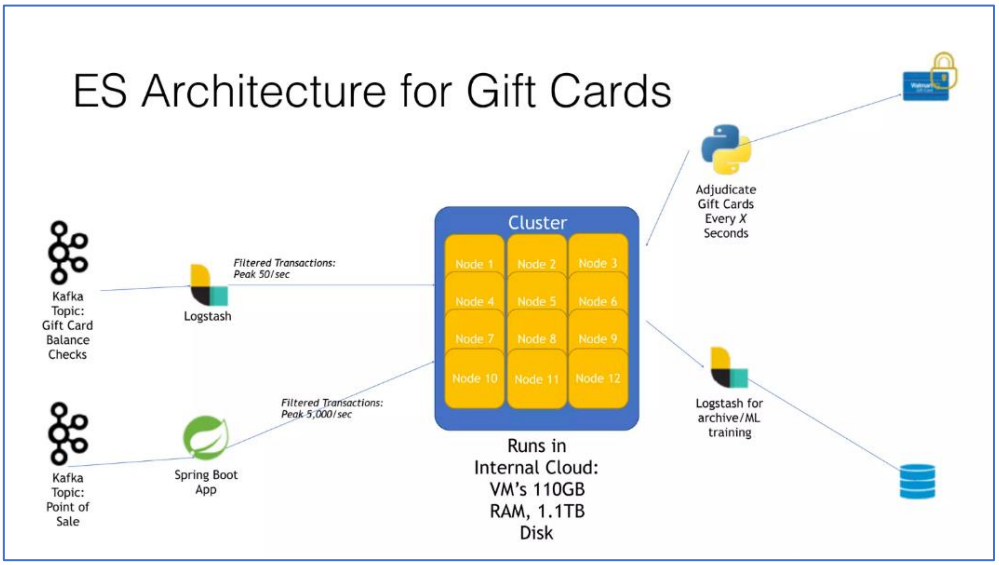
Elastic Security może być podstawą takiego mechanizmu, aby spełnić te wymagania z wysoką wydajnością, skalowalnością i wyjątkowymi wrażeniami wizualnymi.

Rys. 2 – Wykorzystanie ES w Przemysłowych Systemach Sterowania

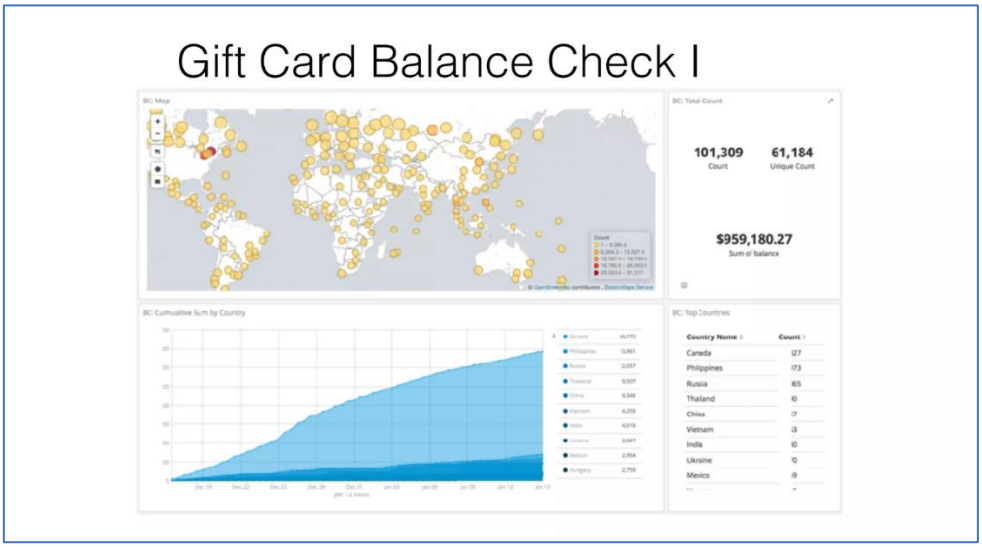
5.3 Walmart

Dzięki Elastic Stack firma Walmart przetworzyła ponad 4 miliardy rekordów metadanych w ciągu ostatnich 5 lat w celu zwalczania oszustw i ochrony klientów. Pobierając adres IP, punkt sprzedaży i inne dane o ruchu w Elasticsearch, zespół Walmart Global Risk Analysis może identyfikować przypadki oszustw w czasie rzeczywistym, w szczególności oszustwa związane z kartami podarunkowymi skierowane do seniorów.

W rezultacie Walmart ochronił miliony dolarów klientów, utrudnia popełnianie oszustw i był w stanie zautomatyzować całodobowy system ochrony bez udziału ludzi.



Rys. 3 – Architektura ES dla Walmart



Rys. 4 – Kibana dashboard dla Walmart

6 Uruchamianie Elasticsearch

6.1 Instalacja i konfiguracja

Wymagania dla instalacji Elasticsearcha są proste – ważne, żeby posiadać Javę w minimalnie wersji 8 (rekomendowana wersja to Oracle JDK wersja 1.8.0_131) oraz żeby Twój system operacyjny znajdował się na Elastic Support Matrix. W przeciwnym wypadku mogą zdarzyć się dziwne i nieprzewidziane sytuacje.

Elasticsearcha można pobrać w wersji standalone albo zainstalować używając repozytoriów. W tym przypadku zainstalujemy Elasticsearcha na systemie Ubuntu 16.04 używając *apt.*

```
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key
add -
echo "deb https://artifacts.elastic.co/packages/6.x/apt stable main" | sudo
tee -a /etc/apt/sources.list.d/elastic-6.x.list
sudo apt-get update
sudo apt-get install elasticsearch
```

Konfiguracja jest wykonywana przy użyciu pliku konfiguracyjnego, którego lokalizacja zależy od systemu operacyjnego. W tym pliku możesz skonfigurować ustawienia ogólne (np. nazwę węzła), ustawienia sieciowe (np. host i port), a także gdzie przechowywane są dane, pamięć, pliki dziennika i inne.

Na przykład, zwłaszcza jeśli instalujesz Elasticsearch w chmurze, dobrym rozwiązaniem jest powiązanie Elasticsearch z prywatnym adresem IP lub hostem lokalnym.

```
sudo vim /etc/elasticsearch/elasticsearch.yml
network.host: "localhost"
http.port:9200
```

6.2 Uruchomienie

Elasticsearch nie uruchomi się automatycznie po instalacji i trzeba będzie go uruchomić ręcznie. Sposób uruchamiania Elasticsearch zależy od konkretnego systemu. W większości systemów Linux i Unix można użyć tego polecenia:

```
sudo service elasticsearch start
```

Aby potwierdzić, że wszystko działa należy wskazać swoją przeglądarką lub narzędziem curl na adres http://127.0.0.1:9200 (alternatywnie http://localhost:9200).

```
{
  "name" : "33QdmXw",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "mTkBe_AlSZGbX-vDIe_vZQ",
  "version" : {
    "number" : "6.1.2",
    "build_hash" : "5b1fea5",
    "build_date" : "2018-01-10T02:35:59.208Z",
    "build_snapshot" : false,
    "lucene_version" : "7.1.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

6.3 Tworzenie indeksu

Indeksowanie to proces dodawania danych do Elasticsearch. Dzieje się tak dlatego, że kiedy wprowadzasz dane do Elasticsearch, dane są umieszczane w indeksach Apache Lucene. Ma to sens, ponieważ Elasticsearch używa indeksów Lucene do przechowywania i pobierania swoich danych.

Elasticsearch zachowuje się jak interfejs API REST, więc możesz użyć API lub metody, aby dodać do niego dane.

```
curl -XPOST 'localhost:9200/logs/my_app' -H 'Content-Type: application/json' -
d'
{
        "timestamp": "2018-01-24 12:34:56",
        "message": "User logged in",
        "user_id": 4,
        "admin": false
}
'
```

```
curl -X PUT 'localhost:9200/app/users/4' -H 'Content-Type: application/json' -
d '
{
  "id": 4,
  "username": "john",
  "last_login": "2018-01-25 12:34:56"
}
'
```

Odpowiedź od bazy danych jest następująca:

```
{"_index":"logs","_type":"my_app","_id":"ZsWdJ2EBir6MIbMWSMyF","_version":1,"r
esult":"created","_shards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"
_primary_term":1}
{"_index":"app","_type":"users","_id":"4","_version":1,"result":"created","_sh
ards":{"total":2,"successful":1,"failed":0},"_seq_no":0,"_primary_term":1}
```

Dane do dokumentu przesyłane są jako obiekt JSON. Być może zastanawiasz się, jak możemy indeksować dane bez definiowania struktury danych. Z Elasticsearch, podobnie jak z każdą inną bazą danych NoSQL, nie ma potrzeby wcześniejszego definiowania struktury danych. Aby jednak zapewnić optymalną wydajność, można zdefiniować mapowania Elasticsearch zgodnie z typami danych.

Jeśli korzystasz z któregokolwiek z dostawców Beats (np. Filebeat lub Metricbeat) lub Logstash, te części stosu ELK automatycznie utworzą indeksy.

Aby zobaczyć listę swoich indeksów Elasticsearch, używa się:

```
curl -XGET 'localhost:9200/_cat/indices?v&pretty'
health status index
yellow open   logstash-2018.01.23
yellow open   app
yellow open   .kibana
yellow open   logs
```

6.4 Tworzenie klastra

Utrzymywanie klastra Elasticsearch może być czasochłonne, zwłaszcza jeśli tworzy się DIY ELK. Jednak biorąc pod uwagę potężne możliwości wyszukiwania i analizy Elasticsearch, takie klastry są niezbędne.

Klastry Elasticsearch grupują wiele węzłów i/lub wystąpień Elasticsearch. Głównym celem takiego grupowania jest podział zadań, wyszukiwanie i indeksowanie klastra w jego węzłach. Opcje węzłów obejmują węzły danych, węzły główne, węzły klienta i węzły pozyskiwania.

Instalowanie węzłów może obejmować wiele konfiguracji, ale oto podstawowa instalacja węzła klastra Elasticsearch:

```
# Instalacja javy
sudo apt-get install default-jre
# Dodanie klucza elastic oraz update systemu
wget -qO - https://artifacts.elastic.co/GPG-KEY-elasticsearch | sudo apt-key
add -
sudo apt-get update && apt-get install elasticsearch
```

Następnie należy utworzyć i/lub skonfigurować własny plik konfiguracyjny każdego węzła Elasticsearch. Po uruchomieniu Elasticsearch możemy sprawdzić stan klastra. Odpowiedzi będą wyglądać mniej więcej tak:

```
{
        "cluster_name" : "elasticsearch-cluster-demo",
        "compressed_size_in_bytes" : 255,
        "version" : 7,
        "state_uuid" : "50m3ranD0m54a531D",
        "master_node" : "IwEK2o1-Ss6mtx50MripkA",
        "blocks" : { },
        "nodes" : {
          "m4-aw350m3-n0D3" : {
            "name" : "es-node-1",
            "ephemeral_id" : "x50m33F3mr--A11DnuM83r",
            "transport_address" : "172.31.50.123:9200",
            "attributes" : { }
          }
        }
}
```

Kondycja klastra Elasticsearch będzie następna na liście. Okresowo należy sprawdzać kondycję klastra za pomocą następującego wywołania interfejsu API:

```
curl -X GET
"localhost:9200/_cluster/health?wait_for_status=yellow&local=false&level=shard
s&pretty"
```

6.5 Przykładowe zapytania

Po zaindeksowaniu danych w Elasticsearch można rozpocząć ich wyszukiwanie i analizę. Najprostszym zapytaniem, jakie można wykonać, jest pobranie pojedynczego elementu.

Po raz kolejny za pośrednictwem API REST Elasticsearch używamy *GET*:

```
curl -XGET 'localhost:9200/app/users/4?pretty'
```

Odpowiedź jest następująca:

```
{
        "_index" : "app",
        "_type" : "users",
        "_id" : "4",
        "_version" : 1,
        "found" : true,
        "_source" : {
                "id" : 4,
                "username" : "john",
                "last_login" : "2018-01-25 12:34:56"
        }
}
```

Pola zaczynające się od podkreślenia to wszystkie „meta” pola wyniku. Obiekt jest oryginalnym dokumentem, który został zindeksowany.

Używamy również *GET* do wyszukiwania, wywołując endpointy:

```
curl -XGET 'localhost:9200/_search?q=logged'
{"took":173,"timed_out":false,"_shards":{"total":16,"successful":16,"skipped":
0,"failed":0},"hits":{"total":1,"max_score":0.2876821,"hits":[{"_index":"logs"
,"_type":"my_app","_id":"ZsWdJ2EBir6MIbMWSMyF","_score":0.2876821,"_source":
{
    "timestamp": "2018-01-24 12:34:56",
    "message": "User logged in",
    "user_id": 4,
    "admin": false
}
}]}}
```

Wynik zawiera szereg dodatkowych pól opisujących zarówno wyszukiwanie, jak i wynik. Oto krótkie podsumowanie:

- *took* – czas wyszukiwania w milisekundach,
- *timed_out* – jeśli upłynął limit czasu wyszukiwania,
- *_shards* – liczba przeszukanych fragmentów Lucene oraz ich wskaźniki powodzenia i niepowodzenia,
- *hits* - rzeczywiste wyniki wraz z metainformacjami o wynikach.

Wyszukiwanie, które przeprowadziłem powyżej, jest znane jako wyszukiwanie URI i jest najprostszym sposobem na zapytanie Elasticsearch. Podając tylko słowo, ES przeszuka wszystkie pola wszystkich dokumentów w poszukiwaniu tego słowa.

6.6 Słowa kluczowe

Elasticsearch zapewnia zaawansowany zestaw opcji do wykonywania zapytań dotyczących dokumentów w różnych przypadkach użycia, dlatego warto wiedzieć, które zapytanie zastosować do określonej sprawy. Każde zapytanie omówione poniżej jest podzielone na 2 typy:

- Zapytania strukturalne - zapytania używane do pobierania uporządkowanych danych, takich jak daty, liczby, kody PIN itp.
- Zapytania pełnotekstowe – zapytania używane do wykonywania kwerend dotyczących zwykłego tekstu.

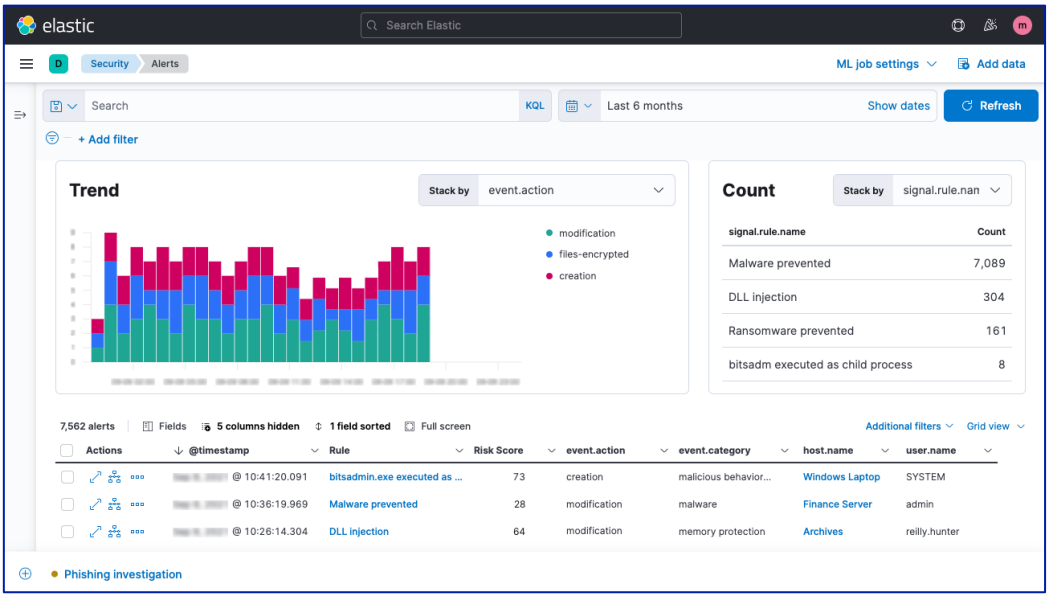
Kategoria	Typ	Kryteria dopasowania
match	Zapytanie pełnotekstowe	Pasuje, jeśli w polu występuje dowolne słowo kluczowe wyszukiwania (analiza odbywa się również na słowach kluczowych wyszukiwania)
multi-match	Zapytanie pełnotekstowe	Aby zastosować zapytanie dopasowujące do wielu pól
match_phrase	Zapytanie pełnotekstowe	Spróbuje dopasować dokładne wyrażenie, w tej samej kolejności
match_phrase_prefix	Zapytanie pełnotekstowe	Spróbuje dopasować dokładną frazę w kolejności, ale ostatni termin będzie pasował jako przedrostek
term	Zapytanie strukturalne	Zapytanie jest stosowane do wygenerowanych tokenów Ponieważ nie jest przeprowadzana żadna analiza, słowo kluczowe jest wyszukiwane jako dopasowanie ścisłe
exists	Zapytanie strukturalne	Zwraca dokumenty zawierające indeksowaną wartość dla pola
range	Zapytanie strukturalne	Zwraca dokumenty zawierające wartości z określonego zakresu określonego w zastosowanym polu
ids	Zapytanie strukturalne	Zwraca dokumenty, które mają określone identyfikatory dokumentów
prefix	Zapytanie strukturalne	Wyszukaj dokładny termin (w tym wielkość liter) na początku słowa
wildcard	Zapytanie strukturalne	Dopasowuje wszystkie terminy z podanym wzorcem wieloznacznym

ES posiada znacznie więcej słów kluczowych w ramach swojej funkcjonalności, ale nie ma potrzeby przechodzenia po wszystkich.

7 Moje doświadczenie

Sercem każdego systemu SIEM (Security Information and Event Management) są dane. Dużo danych. Niezależnie od tego, czy chodzi o serwery, zapory ogniowe, bazy danych czy routery sieciowe — logi dostarczają analitykom surowego materiału do uzyskiwania wglądu w zdarzenia zachodzące w środowisku IT.

Z Elasticsearchem miałem do czynienia w roku 2022 w Warszawskiej firmie zajmującej się szkoleniem pracowników z zakresu cyberbezpieczeństwa. Firma ta chciała w tamtym momencie postawić architekturę bazodanową, która zbierałaby i zajmowała się logami z urządzeń końcowych pracowników. Technologia, która została wybrana, to Elasticsearch razem z całym stackiem ELK. Postawienie tej infrastruktury zajęło pełne dwa miesiące (ze względu na wiele problemów wynikających z niedostępnych zasobów), a następna konfiguracja systemu SIEM – kolejny miesiąc. Po tym okresie czasu informacje zbierane z urządzeń końcowych przechodziły dodatkowy etap „enrichmentu", a następnie trafiały do bazy danych elasticsearch. Na podstawie tych informacji generowane były alerty oraz incydenty w Kibanie.



Rys. 5 – Kibana wykorzystywana jako SIEM

Elastic Security łączy Elastic SIEM, którego silnik wykrywania automatyzuje wykrywanie zagrożeń, dzięki czemu można szybko badać zagrożenia i na nie reagować, oraz Endpoint Security w jedno rozwiązanie, które łączy zapobieganie, wykrywanie i reagowanie w całej sieci.

8 Inne rozwiązania bazodanowe

W trakcie prezentacji, dla studentów 3 roku Informatyki na Politechnice Gdańskiej, przedstawiłem 3 rozwiązania bazodanowe, które, tak samo jak elasticsearch, wyróżniają się pewnymi charakterystycznymi funkcjonalnościami.

- Meilisearch - Rozwiązanie open source dostępne dla wszystkich i wymagające bardzo niewielkiej konfiguracji do zainstalowania, a jednocześnie wysoce konfigurowalne. Obsługuje Błyskawiczne wyszukiwanie, w tym obsługę literówek, filtry, niestandardowe rankingi, etc. Odpowiedzi z bazy danych są zwracane w mniej niż 50 milisekund. Wyróżnia się tym, że jest open source (pomimo bycia komercyjnym) i, z założenia, ma być prostszy we wdrożeniu i utrzymaniu niż jego konkurenci.
- Sonic - może być używany jako prosta alternatywa dla bardzo ciężkich i w pełni funkcjonalnych backendów wyszukiwania, takich jak Elasticsearch w niektórych przypadkach użycia. Zdolny do normalizacji zapytań w języku naturalnym, automatycznego uzupełniania zapytania i dostarczania najbardziej odpowiednich wyników dla zapytania. Deweloperzy naciskają na silną dbałość o wydajność i czystość kodu. System ten jest bezawaryjny, superszybki i w minimalny sposób obciąża zasoby serwera. Według pomiarów, przy dużym obciążeniu, jest to zaledwie 30MB pamięci operacyjnej i bardzo niskie zużycie mocy procesora.

- Manticore Search - Fork technologii open-source zwanej Sphinx Search. Może być używany jako alternatywa dla Elasticsearch zarówno do wyszukiwania pełnotekstowego, jak i analizy danych. Posiada możliwość bezwarunkowego i domyślnego zrównoleglenia zapytania wyszukiwania do wszystkich rdzeni procesora. Szybsze wyszukiwanie i pozyskiwanie danych niż Elasticsearch w konkretnych scenariuszach przedstawionych poniżej.

8.1 Prędkość ManticoreSearch vs Elasticsearch

Nawet jeśli zrobimy tyle odłamków w Elasticsearch, ile jest rdzeni procesora na serwerze, Manticore okazuje się znacznie szybszy, w szczególności: oto test dla 1,7 miliarda dokumentów (dataset dostępny pod linkiem test-taxi), z którego widać, że ogólny Manticore jest 4 razy szybszy niż Elasticsearch.

Query	elasticsearch tuned 32 Fast avg	manticoresearch columnar Fast avg
SELECT * from taxi limit 5	x1.33 (8 ms)	6 ms
SELECT * FROM taxi where match('harlem east') LIMIT 20	x6.88 (8336 ms)	1211 ms
SELECT avg(tip_amount) FROM taxi WHERE tip_amount > 1.5 AND tip_amount < 5	x7.16 (9034 ms)	1261 ms
SELECT avg(total_amount) FROM taxi	x10.71 (23030 ms)	2150 ms
SELECT avg(total_amount) FROM taxi WHERE trip_distance = 5	175 ms	x1.97 (345 ms)
SELECT avg(total_amount), count(*) FROM taxi WHERE trip_distance > 0 AND trip_distance < 5	x11.79 (24699 ms)	2095 ms
SELECT cab_type, count(*) c FROM taxi GROUP BY cab_type order by c desc LIMIT 20	x16.47 (28809 ms)	1749 ms
SELECT count(*) FROM taxi where pickup_ntaname != '0'	x1.32 (818 ms)	618 ms
SELECT count(*) FROM taxi where pickup_ntaname = '0'	8 ms	x39.88 (319 ms)
SELECT count(*) from taxi where pickup_ntaname='Upper West Side'	8 ms	x5.25 (42 ms)
SELECT count(*) FROM taxi WHERE tip_amount = 5	x11.55 (127 ms)	11 ms
SELECT count(*) FROM taxi WHERE tip_amount > 1.5	x4.82 (3260 ms)	676 ms
SELECT passenger_count, avg(total_amount) a FROM taxi GROUP BY passenger_count order by a desc LIMIT 20	x22.83 (81173 ms)	3556 ms
select passenger_count, count(*) c from taxi group by passenger_count order by c desc limit 20	x14.84 (31009 ms)	2089 ms
SELECT pickup_ntaname, count(*) c FROM taxi GROUP BY pickup_ntaname ORDER BY c desc limit 20	x10.42 (22311 ms)	2142 ms
SELECT rain, avg(trip_distance) a FROM taxi GROUP BY rain order by a desc LIMIT 20	x17.18 (71693 ms)	4172 ms
select rain, count(*) c from taxi group by rain order by c desc limit 20	x10.51 (28353 ms)	2698 ms
Arithmetic mean of ratios	x8.87	x3.59
Geometric mean of ratios	x5.67	x1.43

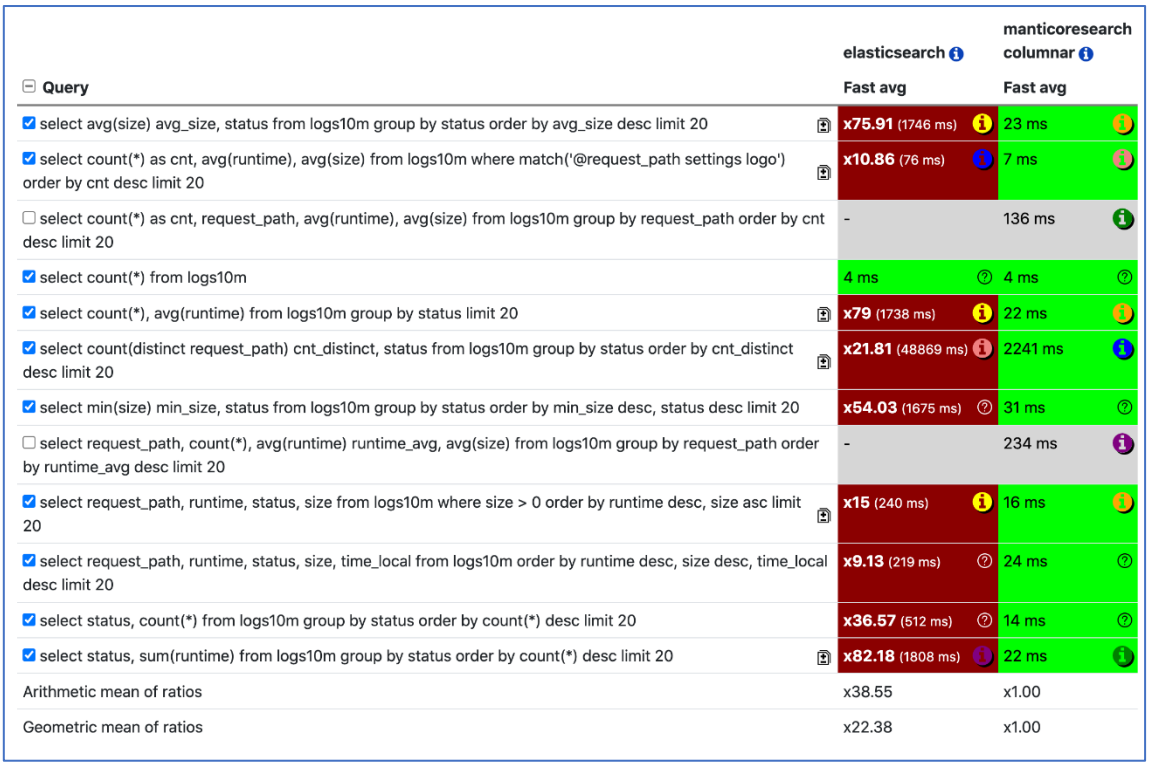
Rys. 6 – Porównanie ES vs ManticoreSearch nr 1

Na następnej stronie znajduje się inny przypadek: brak dużych danych, tylko 1,1 miliona komentarzy z Hacker News (test-hn-small). W tym teście Manticore jest 15x szybszy niż Elasticsearch.

Query	elasticsearch Fast avg	manticoresearch rowwise Fast avg
select * from hn_small order by comment_ranking asc limit 20	x16 (32 ms)	2 ms
select * from hn_small order by comment_ranking asc, story_id asc limit 20	x19 (38 ms)	2 ms
select * from hn_small order by comment_ranking desc limit 20	x16.5 (33 ms)	2 ms
select * from hn_small where match('"elon musk"') limit 20	x13 (13 ms)	1 ms
select * from hn_small where match('abc -google') limit 20	x7 (14 ms)	2 ms
select * from hn_small where match('abc') limit 20	x13 (13 ms)	1 ms
select * from hn_small where match('abc') order by comment_ranking asc limit 20	x13 (13 ms)	1 ms
select * from hn_small where match('abc') order by comment_ranking asc, story_id desc limit 20	x14 (14 ms)	1 ms
select comment_ranking from hn_small order by comment_ranking asc limit 20	x31 (31 ms)	1 ms
select comment_ranking, avg(author_comment_count) avg from hn_small group by comment_ranking order by avg desc, comment_ranking desc limit 20	x76 (380 ms)	5 ms
select comment_ranking, avg(author_comment_count) avg from hn_small where match('google') and comment_ranking > 200 group by comment_ranking order by avg desc, comment_ranking desc limit 20	x4.67 (14 ms)	3 ms
select comment_ranking, avg(author_comment_count) avg from hn_small where match('google') group by comment_ranking order by avg desc, comment_ranking desc limit 20	x4.6 (23 ms)	5 ms
select comment_ranking, avg(author_comment_count+story_comment_count) avg from hn_small group by comment_ranking order by avg desc, comment_ranking desc limit 20	x95.6 (478 ms)	5 ms
select comment_ranking, avg(author_comment_count+story_comment_count) avg from hn_small where comment_ranking < 10 group by comment_ranking order by avg desc, comment_ranking desc limit 20	x69 (207 ms)	3 ms
select comment_ranking, avg(author_comment_count+story_comment_count) avg from hn_small where match('google') and comment_ranking > 200 group by comment_ranking order by avg desc, comment_ranking desc limit 20	x5 (15 ms)	3 ms
select comment_ranking, count(*) from hn_small group by comment_ranking order by count(*) desc limit 20	x30.25 (121 ms)	4 ms
select comment_ranking, story_text from hn_small order by comment_ranking asc limit 20	x16 (32 ms)	2 ms
select count(*) from hn_small	x4 (4 ms)	1 ms
select count(*) from hn_small where comment_ranking > 300 and comment_ranking < 500	5 ms	x1.8 (9 ms)
select count(*) from hn_small where comment_ranking in (100,200)	x5 (5 ms)	1 ms
select count(*) from hn_small where comment_ranking=100	x5 (5 ms)	1 ms
select count(*) from hn_small where comment_ranking=500	x5 (5 ms)	1 ms
select count(*) from hn_small where match('google') and comment_ranking > 200	x8 (8 ms)	1 ms
select story_author, avg(comment_ranking) avg from hn_small group by story_author order by avg desc limit 20	x466.33 (11192 ms)	24 ms
select story_author, count(*) from hn_small group by story_author order by count(*) desc limit 20	x485.09 (10672 ms)	22 ms
select story_id from hn_small order by comment_ranking asc, author_comment_count asc, story_comment_count asc, comment_id asc limit 20	x40 (40 ms)	1 ms
select story_id from hn_small where match('me') order by comment_ranking asc limit 20	x10.5 (21 ms)	2 ms
select story_id, comment_id, comment_ranking, author_comment_count, story_comment_count, story_author, comment_author from hn_small where match('abc') limit 20	x13 (13 ms)	1 ms
Arithmetic mean of ratios	x53.09	x1.03
Geometric mean of ratios	x15.89	x1.02

Rys. 7 – Porównanie ES vs ManticoreSearch nr 2

W kolejnym teście, który wskazuje na Elasticsearch jako standardowe narzędzie do analizy dzienników - 10 milionów logów Nginx (test-logs10m) i różne dość realistyczne zapytania analityczne - Manticore jest tutaj 22 razy szybszy niż Elasticsearch.



Query	elasticsearch Fast avg	manticoresearch columnar Fast avg
select avg(size) avg_size, status from logs10m group by status order by avg_size desc limit 20	x75.91 (1746 ms)	23 ms
select count(*) as cnt, avg(runtime), avg(size) from logs10m where match('@request_path settings logo') order by cnt desc limit 20	x10.86 (76 ms)	7 ms
select count(*) as cnt, request_path, avg(runtime), avg(size) from logs10m group by request_path order by cnt desc limit 20	-	136 ms
select count(*) from logs10m	4 ms	4 ms
select count(*), avg(runtime) from logs10m group by status limit 20	x79 (1738 ms)	22 ms
select count(distinct request_path) cnt_distinct, status from logs10m group by status order by cnt_distinct desc limit 20	x21.81 (48869 ms)	2241 ms
select min(size) min_size, status from logs10m group by status order by min_size desc, status desc limit 20	x54.03 (1675 ms)	31 ms
select request_path, count(*), avg(runtime) runtime_avg, avg(size) from logs10m group by request_path order by runtime_avg desc limit 20	-	234 ms
select request_path, runtime, status, size from logs10m where size > 0 order by runtime desc, size asc limit 20	x15 (240 ms)	16 ms
select request_path, runtime, status, size, time_local from logs10m order by runtime desc, size desc, time_local desc limit 20	x9.13 (219 ms)	24 ms
select status, count(*) from logs10m group by status order by count(*) desc limit 20	x36.57 (512 ms)	14 ms
select status, sum(runtime) from logs10m group by status order by count(*) desc limit 20	x82.18 (1808 ms)	22 ms
Arithmetic mean of ratios	x38.55	x1.00
Geometric mean of ratios	x22.38	x1.00

Rys. 8 – Porównanie ES vs ManticoreSearch nr 3

9 Rozwiązania bazodanowe dla systemów mobilnych

9.1 Elasticsearch dla systemów mobilnych

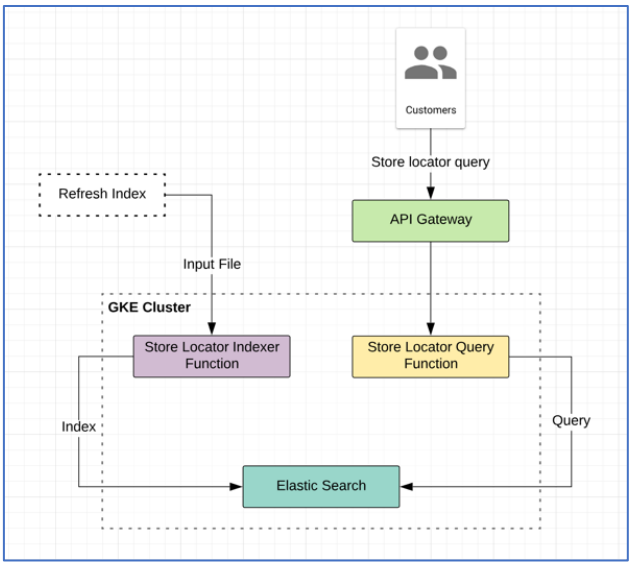
Elasticsearch nigdy nie był i nie jest testowany na urządzeniach mobilnych. Podejrzewam, że system ten byłby zbyt ciężki dla systemów mobilnych. Może to być interesująca rzecz do zabawy, ale prawdopodobnie nie jest to coś, czego powinniśmy szukać w rzeczywistej aplikacji, dla której chcemy włączyć, np. wyszukiwanie offline.

Alternatywą do Elasticsearcha może być Elasticlunr.js. Jest to lekka wyszukiwarka pełnotekstowa w języku JavaScript do wyszukiwania w przeglądarce i wyszukiwania offline. Elasticlunr.js został opracowany w oparciu o Lunr.js, ale jest bardziej „elastyczny” niż lunr.js. Elasticlunr.js zapewnia zmniejszenie czasu zapytania („Query-Time boosting”) i wyszukiwanie po polach (field search). Elasticlunr.js jest trochę podobny do Solr, ale znacznie mniejszy i nie tak jasny, ale także zapewnia elastyczną konfigurację.

9.2 T-Mobile i odległości geograficzne

Mimo tego, że oficjalnie Elasticsearch nie jest przeznaczony na aplikacje mobilne, zawsze znajdą się wyjątki. W tym przypadku wyjątkiem jest T-Mobile, który używa zapytań wyszukiwania odległości geograficznej z Elasticsearch, aby zapewnić doświadczenie „znajdź sklep” w ich aplikacji mobilnej. Obecnie mikrousługa, która napędza to doświadczenie w aplikacji mobilnej, jest rozwijana przy użyciu Vert.x, który jest platformą opartą na Javie, dość powszechnie używaną do tworzenia mikrousług. Niedawno przenieśli oni tę usługę do aplikacji bezserwerowej i zdecydowali się rozwijać ją w golang.

Ponieważ zdecydowaliśmy się oni rozwijać nowe usługi w golang, potrzebowali golang SDK do interakcji z Elasticsearch. T-Mobile nie chciało pisać niskopoziomowych wywołań REST/HTTP do API Elasticsearch i skończyło się na ugodzie z zewnętrznym pakietem SDK.



Rys. 9 – Wysokopoziomowy projekt rozwiązania

Powyższy diagram architektury składa się z dwóch głównych elementów:

- Funkcja Indexer akceptuje kanał CSV wszystkich sklepów i wykonuje upsert do indeksu Elasticsearch.
- Usługa Indexer udostępnia również niektóre operacje administracyjne, takie jak tworzenie indeksu, usuwanie indeksu itp.

Funkcja zapytania przyjmuje wartość szerokości/długości oraz promień i zwraca listę sklepów.