Hämtning av gammal historik från sociala medier

Ett arbete om att presentera historik med längre livslängd än ursprungskällan.

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Ett examensarbete av Artur Hellberg

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
E-husets tryckeri
Lunds universitet
Lund 2018

Sammanfattning

Notified är ett mjukvaruföretag som grundades 2010 med idén att skapa en ny mediabevakningstjänst. Idag kan företagets användare med deras omvärldsbevakningsverktyg (mera känt som Notified) skapa avancerade skräddasydda bevakningar på varumärken, produkter, branscher, konkurrenter mm. Bevakningarna täcker över 50 miljoner källor i hela världen. Därmed kan användarna alltid vara globalt uppdaterade med de senaste nyheterna inom sina intresseområden.

När en användare skapar en bevakning hämtas historik från medier, såsom Facebook, Twitter och YouTube. Notifieds sökresultat är därmed begränsad av livslängden av källornas historik. Företaget vill gå ifrån beroendet av hur källorna har hand om sin historik och har därför sedan en tid tillbaka sparat utdrag av användarnas sökresultat i Notifieds databas. Denna historik sparas även efter den har blivit raderad från ursprungskällan. Dock använder inte Notified den sparade historiken i dagsläget.

Målet med examensarbetet är att ge verktyget Notified tillgång till den sparade historiken i Notifieds databas. Syftet är att med historiken ge användarna mer kompletta sökresultat i framtida sökningar.

Under examensarbetet analyserades två alternativ som kunde uppfylla examensarbetets mål. Lösningen blev att använda en mellanhand som kunde kommunicera direkt med Notified, samt en mjukvara som kunde uppdatera mellanhanden med historik. Mellanhanden är ett verktyg kallad Elasticsearch medan mjukvaran som hämtar historik från Notifieds databas kallas Logstash. Ett API var från examensarbetets början tänkt att ingå i lösningen, men togs sedan bort efter en analys av dess ansvar.

Nyckelord: Notified, sociala medier, Elasticsearch.

Abstract

Notified is a software company founded in 2010 with the idea of creating a new media surveillance service. Today the company's users can use their monitoring tool (also known as Notified) to create advanced customized monitoring on trademarks, products, branches, competitors and more. The surveillance covers over 50 million sources all over the world. Their users can thus always be globally updated with the latest news in their area of interests.

When a user creates a monitoring history is fetched from media, such as Facebook, Twitter and YouTube. The results of Notified are thus limited by the lifespan of the history of the sources. The company wants to go away from the dependency of how the sources handles their history and has thus since some time back saved extracts of the history from the user's results. This history is saved even after it has been deleted from the original source. However Notified currently isn't using this saved history.

The goal of this Bachelor's Thesis is to give tool Notified access to the history in Notifieds database. The purpose is to give the users a more complete result in future searches.

Two alternatives were analyzed during the Bachelor's Thesis that could fulfill its goal. The solution became a intermediator that could communicate directly with Notified and a software that could update the intermediator with history. The intermediator is a tool called Elasticsearch whilst the software that fetches history from Notified's database is called Logstash. An API was intended to be included in the solution from the beginning of the thesis, but was removed after an analysis of its responsibilities.

Keywords: Notified, social media, Elasticsearch.

Innehållsförteckning

1. Inledning

I detta kapitel förklaras bakgrunden bakom examensarbetet såsom företaget, sociala medier, syfte, mål, motivering, problemställning och avgränsningar.

1.1 Om företaget och Notified

Företaget Notified är ett mjukvaruföretag som grundades 2010 med idén att skapa en ny mediabevakningstjänst.[14] Företaget har idag över 25 anställda och erbjuder sina kunder möjligheten att skapa avancerade skräddasydda bevakningar på varumärken, produkter, branscher, konkurrenter mm. Detta är möjligt med deras omvärldsbevakningsverktyg (mera känt som Notified) som täcker över 50 miljoner källor i hela världen. Därmed kan användare alltid vara globalt uppdaterade med de senaste nyheterna inom sina intresseområden. Några av företagets kunder är SVT, ABB, Spotify och SF Bio.

När en användare lägger upp en bevakning hämtas historik från sociala medier och massmedia, såsom Facebook, Twitter och Youtube. Notifieds sökresultat är därmed beroende av källornas historik. Vissa mediers hantering av historik (exempelvis Twitter) gör att bara en veckas historik finns tillgänglig. Då företaget Notified vill gå ifrån beroendet av mediers hantering av historik så sparas resultaten från användarnas sökningar i Notifieds databas. På grund av upphovsrättslagen får hela artiklar ej sparas, utan endast utdrag.

Dock använder inte Notified den sparade historiken i dagsläget. Företaget är intresserat av att använda historiken för att förbättra sökresultaten för sina användare. Detta för att kunna presentera mer fullständiga resultat vid sökningar som går långt tillbaka i tiden, även om informationen raderats från ursprungskällan. Därmed kan Notified ge mer historik till sina användare i framtida sökningar än om man skulle söka direkt från källorna.

1.2 Sociala medier

Enligt NE.se [20] är definitionen av sociala medier följande:

> *Sociala medier är ett samlingsnamn på kommunikationskanaler som tillåter användare att kommunicera direkt med varandra genom exempelvis text, bild eller ljud.*

Sociala medier används idag i stor utsträckning av allmänheten för att kommunicera med bekanta och utbyta information. Några exempel på sociala medier är sociala nätverk som Facebook och Twitter, media-nätverk som Youtube och diskussionsforum som Reddit. Till skillnad från massmedia bygger sociala medier på självproducerat innehåll och åsikter om dessa, medan massmedier fokuserar på att förmedla en neutral bild av händelser runt om i världen.

Sociala medier är en viktig källa för marknadsföring då allmänhetens åsikter väger tungt för företagens rykte och deras försäljningssiffror. Utvecklare kan ta del av användarnas åsikter och använda dessa för framtida uppdateringar. Företag kan förmedla information till konsumenter och besvara frågor. Genom att bygga ett förtroende kan företag stärka sitt varumärke och sin framtid.

1.3 Syfte

För att undvika att Notifieds användare förlorar historik med tiden vill företaget utnyttja den insamlade informationen från tidigare bevakningar i deras databas. Detta ska, om tidigare sökningar gjorts med samma och/eller liknande nyckelord, ge användarna en mer fullständig sökning än om de sökt direkt från källorna. Många av företagets kunder använder Notified som ett verktyg i marknadsföring där information är väldigt viktigt, och därmed är en mer komplett sökning något Notified kan främja nuvarande användare och erbjuda framtida kunder.

1.4 Mål med examensarbetet

När en användare skapar en bevakning ska Notified inkludera historik från tidigare sökningar. Historiken ska presenteras tillsammans med resultat från alla andra källor. Om detta inte är möjligt (t.ex. av tekniska orsaker) ska en analys göras om vad som behövs för att möjliggöra målet. Förfrågan om historik ska vara likt eller av samma format som idag skickas från Notified till 3:e parters API:er.

1.5 Motivering till examensarbetet

Information i medier kan försvinna p.g.a. rensningar och förinställda livslängder hos tredje parter, trots deras popularitet och/eller betydelse. Företaget Notified vill bevara denna information för framtida sökningar och samtidigt gå ifrån beroendet av hur tredje parter hanterar sin historik. På så sätt kan både företag och privatpersoner göra uppföljningar av längre evenemang utan att oroa sig över att informationen blir bortsållad med tiden. Dessutom ges möjligheten att se hur historiken ändrats med tiden, vilket är nytt för företaget. Genom att spara och bevara denna information förblir företagets kunder välinformerade genom sina sökningar, oberoende av tid och ursprungskällors hantering av historik.

Examensarbetet valdes p.g.a. möjligheten att arbeta med databaser och API:er, en värdefull erfarenhet som har många användningsområden. Att utveckla examensarbetet för en tjänst som påverkar både företaget och deras kunder var också en orsak.

1.6 Problemformuleringar

Nedan följer några av de problemställningar som uppstod under examensarbetet samt en beskrivning av dessa.

Hur får man gamla sökresultat att dyka upp i nya, unika sökningar om nyckelorden inte är identiska men ämnet är relevant?

Notifieds bevakningar skapas av användarens inmatningar på nyckelord. Dock finns ingen stavningskontroll eller metod för att identifiera synonymer/ relevanta ord till användarens önskemål vilket kan leda till att relevant information exkluderas. Hur löser man detta, eller ska man utgå ifrån att användarens inmatning är felfri?

Hur undviks dubblering av resultat?

Användare som lägger upp bevakningar kan besväras av flera sökresultat som hänvisar till samma artikel, utdrag och/eller citat fastän de kommer från olika källor. För att undvika att Notifieds databas presenterar historik som redan existerar måste en metod användas som kan jämföra artiklar, utdrag och/eller citat innan de presenteras för användaren.

Kan man koppla Amazon Aurora-servern till resultatet?

Notified har gått över till en Cloud-server kallad Amazon Aurora. Examensarbetet kommer göras med en PostgreSQL-databas, en kopia av Notifieds tidigare databas. Därmed måste resultatet kunna koppla till Notifieds nya databas i slutet av examensarbetet. Om detta inte är möjligt (t.ex. av tekniska orsaker) måste en analys göras om behövs för att möjliggöra det.

Är ett API nödvändigt för en fungerande lösning?

Det första lösningsförslaget (se avsnitt *3.3 Förstudier* samt *figur 6*) var att ha ett API som mellanhand till Notified och dess databas. Men är det nödvändig eller finns det ett bättre alternativ?

1.7 Avgränsningar

Nedan listas avgränsningarna på examensarbetet (ses även i avsnitt 9.2 Krav från företaget Notified angående lösning) samt deras motivering, överenskommet med företaget.

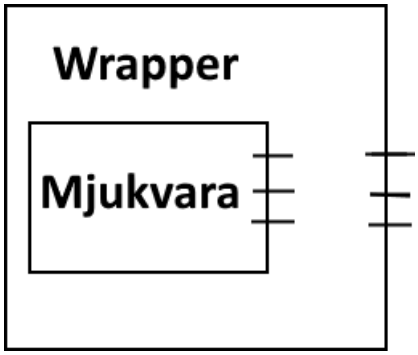
- *Examensarbetet involverar enbart lösningar för operativsystemet Windows.* Detta eftersom Windows är en väsentlig del av Notifieds arbetsmiljö samt att det underlättar framtida vidareutveckling.
- *API:et som söker igenom Notifieds databas ska vara ett booleskt sök-API.* Detta eftersom Notified använder booleska operationer för att fråga 3:e parter om information utifrån användarnas skräddarsydda sökningar.
- *Nuvarande datastrukturen i Notifieds PostgreSQL-databas ska behållas, dock kan tillägg i form av datatabeller förekomma.* Eftersom företagets tjänster är beroende av databasen och ändringar kan medföra säkerhetsrisker ska inga ändringar göras utan företagets godkännande. Helst ska inga ändringar göras alls.

2. Teknisk Bakgrund

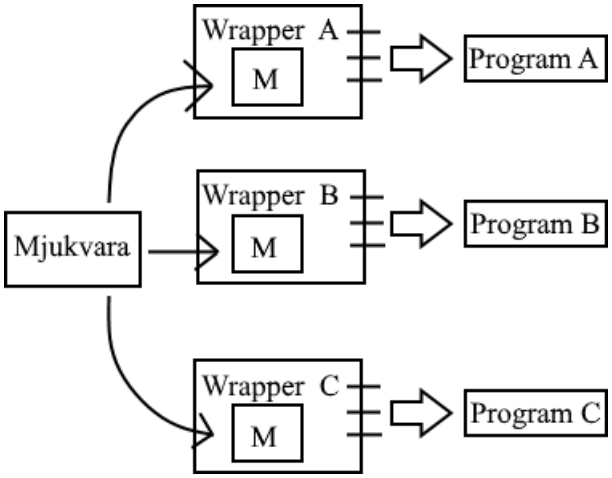
Detta kapitel presenterar alla programmeringsspråk samt verktyg i både mjukvara och hårdvara som använts under examensarbetet.

2.1 Wrapper (mjukvara)

En wrapper är en mjukvara som används för att omförsluta en mjukvara (se *figur 1*) [21]. Wrappers har flera användningsområden, såsom att göra den omförslutna mjukvaran kompatibel med de parter den kommunicerar med eller lägga till ytterligare funktionalitet utan att ändra i mjukvaran. Detta är fördelaktigt för mjukvara som kommunicerar med många andra mjukvaror, då en anpassad wrapper kan skapas för mjukvaran den kommunicerar med (se *figur 2*). Därmed undviker utvecklarna att göra en version av mjukvaran för varje program den kommunicerar med, vilket gör uppdateringar enklare.



Figur 1: Exempel på en wrapper som utvidgar mjukvarans funktionalitet utan att ändra i dess program.



Figur 2: Exempel på hur Wrappers kan användas för att anpassa en mjukvara för flera program med hjälp av wrappers.

2.2 REST

REST är ett designkoncept för hantering och strukturering av information. REST står för *Representational State Transfer* och bygger på sex principer: *Uniform Interface*, *Cachable*, *Stateless*, *Layered System*, *Client-Server* och *Code on demand*. [1]

Uniform Interface innebär att det finns ett färdigt gränssnitt för kommunikationen mellan klienten och servern: HTTP och URI.

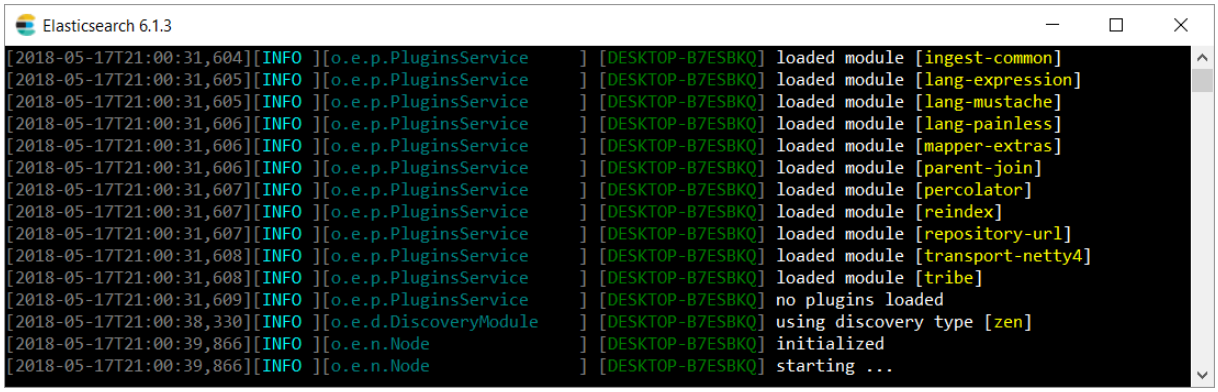
Cachable innebär att informationen måste vara lagringsbar. Klienter får all information av intresse för att kunna bearbeta och/eller analysera informationen på egen hand. Informationen bearbetas därmed från klientens sida, inte serverns.
Stateless innebär att varje meddelande mellan server och klienten har tillräcklig med information för att servern ska förstå klientens förfrågan. Servern ska inte vara i ett särskilt tillstånd eller behöva memorera tidigare förfrågningar för att kunna förstå en ny förfrågan.
Layered System innebär att man utgår ifrån att kommunikationen sker via mellanhänder t.ex. API:er, wrappers, routrar eller annan mjukvara. Detta för att hantera svar som inte kommer från klienten eller servern, såsom felmeddelanden eller begäran av mer information.
Client-Server innebär att man ska kunna hantera oväntade händelser. Felmeddelanden, avbruten kommunikation eller mottagna meddelanden som inte är kompletta är tre exempel. Genom att hantera oväntade fall blir mjukvaran säkrare och kan informera när något inte står rätt till.
Code on demand innebär att server kan (temporärt) skicka beräkningar till klienten för exekvering istället för att göra det själv. Till skillnad från tidigare koncept är denna valbar.

REST används för att få mjukvaran att kommunicera via HTTP, vilket är vida använt på Internet. REST är resursbaserat, vilket innebär att istället för att bearbeta klienters förfrågningar med metoder så förses klienten med alla resurser som behövs. REST används huvudsakligen för design av webbapplikationer.

2.3 Elasticsearch

Elasticsearch är ett REST-verktyg (se avsnitt *2.2 REST*) utvecklat av företaget Elastic. [9] Verktyget kan lagra och sortera information samt kan anropas med många programmeringsspråk och inmatningar, t.ex. SQL och booleska operationer via HTTP. Beroende på hur användaren sätter upp Elasticsearch kan verktyget agera sökmotor och/eller databas. Skillnaden är att sökmotorer inte lagrar någon egen information medan databaser strukturellt lagrar information.

Elasticsearch kan installeras med en WIN-fin för Windows-användare. Denna installation skapar en genväg på datorns skrivbord till skillnad från andra installationsfiler. Vid start öppnas en kommandotolk (se *figur 3*) som inte kan användas för inmatningar, utan presenterar endast server-status och händelser.



Figur 3: Elasticsearchs kommandotolk vid start av program.

Elasticsearch 6.1.3 användes under examensarbetet som en databas för att lagra specifik information som Notified använder till sina medie-sökningar. Med andra ord agerade Elasticsearch som en bibliotekarie: När Notified frågar returneras informationen baserade på de sökord som ges. Då Elasticsearch inte har funktionaliteten att hämta information på egen hand kompletteras programmet med Logstash (se avsnitt *2.4 Logstash*).

2.4 Logstash

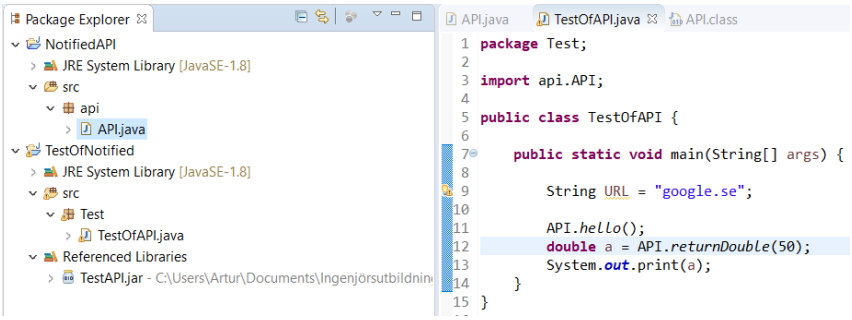
Logstash är en insamlingsmotor som hämtar information från databaser, filtrerar och/eller konverterar informationen och slutligen exporterar till andra databaser beroende på användarens önskemål. Logstash är av öppen källkod utvecklat av företaget Elastic. [10]

Logstash 6.1.3 användes under examensarbetet för att exportera information från Notifieds postgreSQL-databas till Elasticsearch (se *figur 9* avsnitt *3.5.2 Steg 2 – Logstash* samt avsnitt *2.3 Elasticsearch*). Hämtningen gjordes genom skapandet av en konfigurationsfil (logstash.conf) som innehöll bl.a. ett sparat SQL-kommando som efterfrågade de tabeller som var av intresse. Tabellerna levererades i önskat format (JSON) till Elasticsearch. Se *figur 8* i avsnitt *3.5.2 Steg 2 - Logstash* för bild av konfigurationsfilen.

2.5 Eclipse

Eclipse är ett IDE-verktyg framtaget av Eclipse Foundation för utveckling och underhåll av mjukvara.[6][7] Genom skapande av projekt, paket och klasser kan användaren designa program med en eller flera programmeringsspråk. Eclipse kan även konvertera och exportera mjukvara till andra miljöer samt importera paket till den egna miljön.

Eclipse Oxygen.2 version 4.7.2 användes vid utveckling och testning av API:et under examensarbetet (se *figur 4*). API:et utvecklades med programmeringsspråket Java. Dess stöd för felsökning och detektering av buggar i mjukvara samt flera års erfarenhet med verktyget för programmering i Java gjorde Eclipse till ett självklart val. Tyvärr har Eclipse svårigheter att hantera tecken utanför det engelska alfabetet såsom kinesiska, vilket begränsade förståelsen vid presentation av resultat hämtade från Notifieds databas då alla okända tecken byts ut till ⍰.



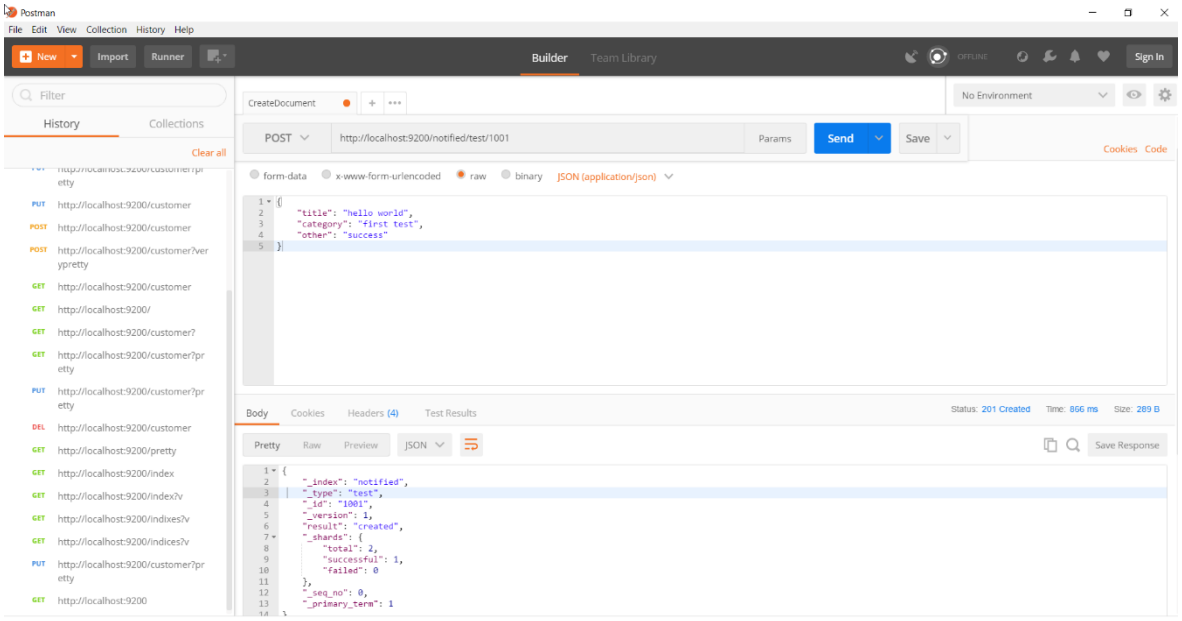
Figur 4: API:et under utveckling i Eclipse utvecklingsmiljö.

2.6 Postman

Postman en utvecklingsmiljö framtagen av Postdot Technologies Inc för API:er och HTTP-förfrågningar.[8] Postman 1.4.2.0 användes för att hämta, ändra och kontrollera information samt status hos Elasticsearch (se avsnitt *2.3 Elasticsearch*) under utveckling, testning och installation (se *figur 5*). Detta genom att skapa HTTP-förfrågningar som returnerade data, vilket Postman presenterade. Fler funktioner fanns tillgängliga, men de ovannämnda var de som användes under examensarbetet.

Postman valdes baserat på tidigare erfarenheter med programmet, rekommendationer från privatpersoner med programmeringskunskaper och för

att den kunde hantera vissa tecken som Eclipse inte kunde presentera, såsom kinesisk skrift (se avsnitt *2.5 Eclipse*). Postman tog mindre exekveringstid vid frågor till Notifieds databas och Elasticsearch jämfört med java-klasserna i Eclipse.



Figur 5: Postman vid test av Elasticsearch.

2.7 PostgreSQL & PgAdmin4

PostgreSQL är ett objekt-relationellt databassystem av öppen källkod. [11] [13] Med objekt-relationellt menas en databashanterare som kompletterats med en objektorienterad datamodell (klasser, arv, metoder). Detta system används både av Notifieds tidigare och nya databas, Amazon Aurora.

PgAdmin4 är en utvecklingsplattform för postgreSQL. Med detta verktyg av öppen källkod kan användaren skapa och underhålla databaser. Användaren kan även kommunicera med databaserna med SQL-kommandon via ett fönster för inlägg, läsning, ändring eller radering av information.

PgAdmin4 2.0 användes under examensarbetet för att skapa en kopia av Notifieds databas och läsa av tabellerna av intresse för Notified. SQL-kommandon testades här innan dessa skulle användas av Logstash (se avsnitt *2.4 Logstash*) för hämtning av tabeller. Ingen information i databasen ändrades under examensarbetet.

3. Analys & Metod

Detta kapitel beskriver examensarbetets faser, arbetsmetod, logik och resonemang. Kapitlet innefattar även användningen av verktygen presenterade i Teknisk Bakgrund, analys av lösningsförslag samt utvecklingen av slutresultatet. Avslutningsvis sker en mindre analys av resultatet innan exporteringen till företagets miljö.

3.1 Examensarbetets faser

Examensarbetet skedde i tre faser i följande ordning: *Förstudier*, *Analysfasen* och *Utvecklingsfasen*. Inom varje fas fanns två delfaser: Utveckling och Analys. I och med vald arbetsmetod (se avsnitt *3.2 Arbetsmetod*) växlades arbetssättet mellan utveckling and analys iterativt. Se nedan för beskrivning av utveckling respektive analys i respektive fas.

Förstudier
Under denna fas hämtades information inför examensarbetet. En dialog fördes med kontaktpersonen för Notified om vilka resurser som fanns tillgängliga, vilka krav som ställdes, målet med examensarbetet och information om både företaget och Notified (se avsnitt *9.1 Frågor inför möte med företaget Notified* och *9.2 Krav från företaget Notified angående lösning*).
Problemformuleringar, avgränsningar och syfte byggdes upp (se avsnitt *1.6 Problemformuleringar*, *1.7 Avgränsningar* respektive *1.4 Syfte*). Det första lösningsförslaget att använda ett API som mellanhand togs fram tillsammans med företaget (se *figur 6* i avsnitt *3.3 Förstudier*).

I *Förstudier* bestod utvecklingen av att samla information (se specifika punkter i texten ovan) från företagets kontaktperson. Analysen bestod av att förstå Notifieds nuvarande situation och syfte, analysera det första lösningsförslaget och lära känna tillgängliga resurser. För detaljer av varje uppgift se avsnitt *3.2 Förstudier*.

Analysfasen
Under *Analysfasen* undersöktes två alternativ för att lösa problemet med den första lösningsförslaget (se *figur 6* samt avsnitt *3.3 Förstudier*): Wrappers (se avsnitt *2.1 Wrapper*) eller en mellanhand som kunde förstå Notified. Valet

blev Elasticsearch (se avsnitt *2.3 Elasticsearch*) kompletterat med Logstash (se avsnitt *2.4 Logstash*). Se motivering i avsnitt *3.4 Analysfasen*.

I *Analysfasen* bestod utvecklingen av att söka efter wrappers som kunde uppfylla examensarbetets mål (se avsnitt *1.4 Mål med examensarbete*), delta i en workshop [12] och hitta en komplettering till Elasticsearch. Analysen bestod i sin tur att utvärdera om en wrapper, Elasticsearch och Logstash kunde uppfylla de krav som ställts av företaget (se avsnitt *9.2 Krav från företaget Notified angående lösning*), att de hade funktionalitet som stödde syftet (se avsnitt *1.3 Syfte med examensarbete*) samt att kunskaper fanns för att använda verktygen på rätt sätt. För detaljer av varje uppgift se avsnitt *3.4 Analysfasen.*

Utvecklingsfasen
Utvecklingsfasen delades upp i fyra steg för att utveckla lösningen (se *figur 7* i avsnitt *3.5 Utvecklingsfasen*): Testmiljö, Logstash, API och Testning. I första steget installerades alla nödvändiga verktyg i en testmiljö. I andra steget skapades en konfigurationsfil för Logstash för att överföringen av informationen skulle ske på ett så friktionsfritt sätt som möjligt. I tredje steget utvecklades API:et. I fjärde steget testades Logstash, Elasticsearch och API:et genom att simulera en förfrågan från Notified. Avslutningsvis gjordes en analys av API:et i vilket det framkom att denna inte behövdes. Därmed togs API:et bort.

I *Utvecklingsfasen* bestod utvecklingen av att installera mjukvara, skapa en konfigurationsfil, göra upprepade testningar med Logstash, programmera API:et och testklassen som simulerade Notified. Analysen var en kontroll att mjukvarorna installerats korrekt, att konfigurationsfilen hade korrekta variabler vid överföring, undersöka värdet av antalet rader per cykel för konfigurationsfilen, bekräftning att testningen lyckats samt att analysera om API:et var en nödvändig del av resultatet. För detaljer av varje uppgift se avsnitt *3.5 Utvecklingsfasen.*

3.2 Arbetsmetod

Arbetsmetoden gick ut på att inom varje fas inom examensarbetet (se avsnitt *3.1 Examensarbetets faser*) ha två delfaser: Utveckling och Analys. Utvecklingens syfte är att utföra tilldelad uppgift. Analysens syfte är att analysera resultatet som uppkommit från utvecklingen. Om analysen visar att önskat resultat erhållits går man över till nästa uppgift. I annat fall går man tillbaka till utveckling och använder analysen för att uppnå ett bättre resultat. På så sätt växlar man mellan och itererar delfaserna tills alla uppgifter är avklarade.

Metoden blev inspirerad av iterativa arbetsmetoder av flera anledningar. [15] En av dessa var att metoden är flexibel för förändringar. Detta var viktigt eftersom det inte fanns en färdig lösning, utan något som man fick resonera sig fram till under examensarbetet med öppenhet för bättre alternativ. En annan är att ändringar på krav och resurser kunde uppstå. Detta skulle bli mer kostsamt om arbetsmetoden inte var öppen för förändringar. En sista fördel var att man kunde göra analyser av verktyg, mjukvara och resultat. Eftersom det fanns mycket kontakt med verktyg och mjukvara som man inte hade något tidigare erfarenhet med var det viktigt att försäkra sig om att varje komponent uppfyllde sitt syfte.

Även om varje fas itererade mellan utveckling och analys så behövdes inte alltid en fullständig analys göras. Vid (utvecklings)installation av mjukvara var analysen bekräftelsen från installationsverktyget att mjukvaran installerats korrekt. Vid ett tillfälle gjordes ett undantag. Steg 3- API (se avsnitt *3.5.3 Steg 3- API*) saknade analys eftersom en testning skulle ske efter att utvecklingen var avklarad (se avsnitt *3.5.4 Steg 4 – Testning*). Se avsnitt *3.1 Examensarbetets faser* för var utveckling och analys bestod av under varje fas.

3.3 Förstudier

Under denna fas samlades nödvändig information in för examensarbetet, såsom resurser, krav, mål osv.

Innan utvecklingen kunde påbörjas behövdes information såsom företagets syfte med examensarbetet, krav på slutprodukten samt problem som kunde uppstå. Ett möte bokades med Notifieds kontaktperson Mattias Axelsson där

dessa punkter samt ett antal frågor (se avsnitt *9.1 Frågor inför möte med företaget Notified*) gicks igenom och dokumenterades (se avsnitt *1 Inledning* för resultat från mötet).

En av frågorna som även togs upp på mötet var ”Hur ska jag ta hänsyn till sökresultat som är relevanta men där sökorden inte stämmer överens med innehållet?". Anledningen till frågan är att Notified skickar ut sökorden som användaren matar in till 3:e parter (se avsnitt *1.1 Om företaget och Notified*). Därmed kunde det förväntas att lösningen skulle inte bara koppla Notified med dess databas, men även inkludera all relevant information baserad på användarens sökord och inställningar. Svaret på frågan blev att detta är Notifieds ansvar och inte lösningens. Eftersom Notified ska ha tillgång till hela databasen, oavsett vilka sökord användaren matar in, så är detta inget som man behöver ta hänsyn till.

En annan fråga som kom upp under mötet var ”Hur kan vi undvika dubblering av resultat?”. Anledningen är att om Notified får historik från sin databas som fortfarande existerar i ursprungskällan kommer användaren att visas två identiska resultat. Svaret på frågan blev att detta är Notifieds ansvar och inte lösningens. Notified skickar ut sin förfrågan till alla 3:e parter (och sin egen databas) oberoende av varandra och behövs därmed inte tas hänsyn till.

Målet med examensarbetet är att ge Notified tillgång till sökresultaten från användarnas sökningar i Notifieds databas. För att uppnå detta föreslogs ett API som agerar mellanhand mellan Notified och dess databas (se *figur 6*). Därmed fås en struktur som är enkel att underhålla, övervaka och vidareutveckla, samt utan större förändringar i verktyget eller databasen.

Figur 6. Det första lösningsförslaget. Ett API kan enkelt göra databasen tillgänglig och kan anpassas efter båda parterna.

Dock fanns ett problem. Notifieds databas var av typen postgreSQL, vilket inte accepterar booleska operationer i HTTP som Notified skickar till 3:e parter. Eftersom avgränsningarna specificerade att nuvarande struktur i databasen skulle behållas (se avsnitt *1.7 Avgränsningar*) samt att det inte var

rekommenderat att göra ändringar i verktyget Notified så är detta ett problem som måste lösas mellan databasen och Notified.

Därmed fanns två alternativ: Hitta en översättare som omvandlar booleska operationer till SQL-kommandon för databasen eller exportera all relevant information till en mellanhand som förstår booleska operationer och därefter kan ge den efterfrågade informationen till Notified.

3.4 Analysfasen

Under analysfasen undersöktes de två alternativen presenterade i föregående fas. Tre wrappers analyserades för att lösa problemet med att databasen och Notified inte kunde kommunicera direkt med varandra.

Som presenterat i slutet av föregående fas (se avsnitt *3.3 Förstudier*) fanns två alternativ för att lösa problemet mellan Notified och dess databas: Hitta en översättare som omvandlar booleska operationer till SQL-kommandon eller exportera all relevant information till en mellanhand som förstår booleska operationeri HTTP. Valet blev att undersöka det förstnämnda alternativet först eftersom det inte skulle ändra på lösningsförslaget (se *figur 6* i avsnitt *3.3 Förstudier*) överenskommet med företaget, samt att alternativet verkade enklare att anpassa till Notifieds tidigare och nuvarande (Amazon Aurora) databas än en mellanhand.

En wrapper runt API:et eller databasen skulle göra att Notified skulle förbli oförändrad och samtidigt behålla strukturen på lösningsförslaget. Tre wrappers framstod vid sökning av översättare och analyserades:

- *ZomboDB* är en wrapper till postgreSQL som utvidgar dess funktionalitet och stödjer booleska operationer. [16] Dock så stödjer programmet inte Windows utan är designad för Linux. Eftersom avgränsningarna specifikt beskrev ”Endast Windowslösningar” uteslöts detta alternativ.
- *Postgres-elasticsearch-fdw* är en wrapper som kan översätta booleska operationer i HTTP till SQL-kommandon, samt bygger på programmeringsspråket Python. [17] Dock saknades egna kunskaper i Python för att sätta upp wrappern korrekt och valdes bort.

- *JDBC Input* är en databas-lyssnare som kan hämta information från databaser som stödjer JDBC (Java DataBase Connection) interfacet, vilket även postgreSQL gör. [18] Dock så stödjer inte detta alternativ anrop utan snarare exportering av information från en databas till en annan. Därmed passade inte detta alternativ eftersom Notified skickar förfrågningar som måste besvaras.

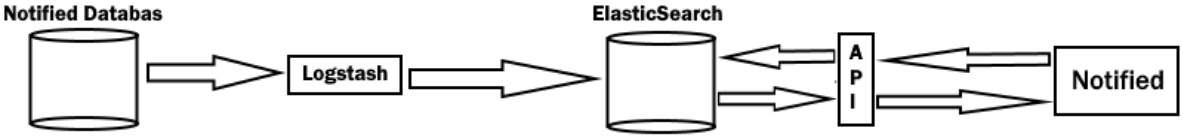
Eftersom inga passande alternativ hittades så hölls ett samtal med företagets anställda angående möjligheten att utveckla en egen översättare. Slutsatsen blev att utvecklingen skulle bli en för stor uppgift för examensarbetets syfte. Dessutom fanns ett alternativ till lösningsalternativet om en mellanhand: Elasticsearch. Eftersom majoriteten av de tidigare alternativen använde eller stödjer Elasticsearch samt att alternativet att exportera all relevant information till en mellanhand som förstår booleska operationer inte analyserats så undersöktes alternativet.

Elasticsearch är ett REST-verktyg som kan lagra och sortera information via kommandon från användare. Se avsnitt *2.3 Elasticsearch* för ytterligare detaljer. Eftersom verktyget accepterar booleska operationer via HTTP var verktyget en passande lösning. Dock behövdes historik hämtas från Notifieds databas, något Elasticsearch inte hade funktionalitet för. Historiken i Elasticsearch behövde uppdateras regelmässigt för att hållas konsistent i.o.m. att Notifieds databas fylls med ny historik med tiden. I ett samtal med en föreläsare under en workshop om Elastic [12] föreslogs Logstash som ett komplement till den funktionalitet som saknades. Eftersom Logstash också utvecklats av Elastic med ändamålet att koppla samman Elasticsearch med andra databaser så valdes verktyget. Se avsnitt *2.4 Logstash* för beskrivning av verktyget.

Det nya lösningsförslaget blev att Logstash ansvarar importering av information till Elasticsearch, som sedan kommunicerar med Notified via ett API. Eftersom Logstash kan välja vilka databastabeller som informationen hämtas från kan Elasticsearch uppdateras med all historik som Notified behöver på en plats. Därmed behöver inte Notified känna till namnen på databastabellerna vid förfrågningar till Elasticsearch.

3.5 Utvecklingsfasen

Utvecklingsfasen tar lösningsförslaget presenterad under Analysfasen (se avsnitt 3.4 Analysfasen) och utvecklar den i 4 steg: Skapa testmiljö, Logstash, API och Testning. Avslutningsvis sker en analys om API:et i resultatet.



Figur 7: Avbildning av lösningsförslaget presenterad i Analysfasen. Logstash hämtar historiken till Elasticsearch, som sedan kommunicerar med Notified via ett API.

Med en bild om hur lösningen skulle se ut från föregående fas (se *figur 7* respektive avsnitt *3.4 Analysfasen*) delades utvecklingen in i fyra steg: *Steg 1 - Skapa testmiljö*, *Steg 2 - Logstash*, *Steg 3 - API* och *Steg 4 - Test av lösning*. Första stegets uppgift är att förbereda en testmiljö för alla verktyg som ska användas. Andra steget ska förbereda Logstash för exportering av historik till Elasticsearch samt testa överföringen. Tredje steget är att skapa API:et. Fjärde och sista steget är att testa lösningen genom ett program som imiterar Notified. Se nedan för genomgång av alla 4 steg.

3.5.1 Steg 1 – Skapa testmiljö

Uppgift: Förbereda en testmiljö. Installera alla nödvändiga verktyg.

En mapp kallad ”Exjobb” skapades för alla hålla alla nödvändiga verktyg i en gemensam testmiljö. Eclipse installerades för utveckling av API:et i programmeringsspråket Java. Postman installerades för att kunna skapa HTTP-förfrågningar och testa Elasticsearch och API:et. PgAdmin 4 installerades för att skapa en kopia av Notifieds databas. Logstash och Elasticsearch installerades för att kunna exportera och spara historiken, tillgänglig för Notified vid förfrågan. Flera HTTP-förfrågningar med Postman bekräftade att både databasen och Elasticsearch var i funktion samt kunde lagra och radera information. Se avsnitt *2 Teknisk Bakgrund* för närmare detaljer om varje verktyg.

3.5.2 Steg 2 - Logstash

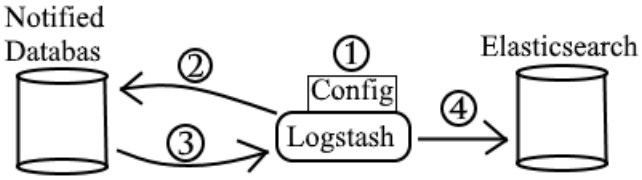
Uppgift: Få Logstash att hämta information från Notifieds databas och exportera informationen till Elasticsearch (som presenterat i figur 7).

En mapp ”test” skapades i Logstash där en textfil kallad ”logstash.conf” konstruerades (se *figur 8* nedan). Denna konfigurationsfil lagrar information om var Logstash kan hitta databaserna som den importerar/exporterar till, vilket format informationen ska vara i, inloggningsuppgifter, SQL-förfrågan mm. Dessa inställningar bestämdes tillsammans med företaget eller resonerades fram. Genom att öppna kommandotolken i mappen ”Logstash 6.1.3” och använda kommandot ”bin\logstash –f test” påbörjade Logstash överföringen. Som demonstrerat i *figur 9* läste Logstash av konfigurationsfilen och följde de skrivna instruktionerna **(1)**. Logstash skickade därefter SQL-förfrågan i konfigurationsfilen till databasens adress **(2)**. Databasen returnerar resultatet till Logstash **(3)** som vidarebefordrar till Elasticsearch **(4)**.

```
input {
        jdbc {
                jdbc_driver_library => "C:\Users\Artur\Documents\Ingenjorsutbildningen\Exjobb\workspace\postgresql-42.2.1.jar"
                jdbc_driver_class => "org.postgresql.Driver"
                jdbc_connection_string => "jdbc:postgresql://localhost/Notified"
                jdbc_user => [redacted]
                jdbc_password => [redacted]
                jdbc_validate_connection => true
                jdbc_paging_enabled => true
                jdbc_page_size => 1200
                clean_run => false
                statement => "SELECT * FROM items LEFT OUTER JOIN itemfulltexts ON items.id=itemfulltexts.itemid ORDER BY items.id"
        }
}

output {
        stdout{ codec => json_lines }
        elasticsearch {
                "hosts" => "127.0.0.1:9200"
                "index" => "migrate-test4"
                document_type => "items"
        }
}
```

Figur 8: Konfigurationsfilen för Logstash. Denna fil innehåller all information som Logstash behöver veta när överföringen av information (i detta fall historik) ska ske.



Figur 9: Vid start läser Logstash av konfigurationsfilen (Config) med all nödvändig information Logstash behöver. Logstash skickar sedan en förfrågan till Notifieds databas och vidarebefordrar resultatet till Elasticsearch.

Dock uppstod ett problem. Vid överföringen fick Logstash slut på minne p.g.a. den enorma mängden data som databasen innehöll. Logstash var därmed oförmögen att hantera all information under en överföring. Lösningen blev att göra en så kallad ”paging”, att repetitivt hämta ett visst antal rader och spara dessa i Elasticsearch innan man hämtar resterande resultat. Men hur många

rader skulle hämtas per cykel? Genom att ta tiden på hämtningen av 10 000 rader så mättes hur antalet rader per cykel påverkade exekveringstiden.

Tabell 1: Exekveringstiden av Logstash hämtning av 10'000 rader med varierande storlek på antalet rader per cykel. Detta för att göra överföringen effektiv och samtidigt inte överbelasta Logstash.

Rader per cykel	Exekveringstid
5	2m 5s
250	1m 22s
500	1m 12s
1000	1m 10s
1200	1m 10s

Som ses i *tabell 1* var tidsvinsterna små efter 500 rader per cykel. Men eftersom varje rad kunde variera i storlek samt att antalet rader per cykel kunde ge större tidsvinster vid större exekveringar (speciellt när databasen hade $3,7 * 10^6$ rader) så behölls storleken på 1200 rader per cykel.

Med Postman kunde man bekräfta att Logstash hade fört över rätt antal rader till Elasticsearch samt att informationen var tillgänglig via anrop.

3.5.3 Steg 3- API

Uppgift: Skapa ett API och koppla denna till Elasticsearch.

Ett java-projekt skapades i Eclipse kallad ”NotifiedAPI”. I detta projekt skapades två klass-paket: ”API” och ”LocalTest”. Anledningen till att test-klassen var i ett separat paket var för att simulera en exportering av API:et (se avsnitt *3.5.4 Test av lösning*). I API-paketet skapades en java-klass kallad ”ES_API” (se *figur 10* nedan), vilket blev API:et för att koppla samman Elasticsearch och Notified.

```
ES_API.java   LocalTestOfES_API.java
 1 package api;
 2 import java.io.BufferedReader;
 7
 8 public class ES_API {
 9
10     private static HttpURLConnection connection = null;
11     private static final String address = "http://localhost:9200/";
12     private static final String mapping = "migrate-test1";
13     private static URL url = null;
14     private static boolean open = false;
15
16     private static boolean openConnection(){
28
29     private static void closeConnection() {
37
38     private static boolean setQuestion(String httpQuestion) {
52
53     /**Opens a connection to the database and checks if it replies.
54      *
55      * @return Returns true if a connection was made with a response, else false.
56      */
57     public static boolean testConnection(){
77
78     /**Executes a HTTP-request and returns the result from the Notified database.
79      *
80      * @param httpRequest The HTTP-request. The address and mapping is set.
81      * @return Returns a String with the results of the database.
82      */
83     public static String executeHTTPRequest(String httpRequest) {
106 }
```

Figur 10: Java-klassen ”ES_API” i Eclipse.

För att få API:et att koppla upp till Elasticsearch så implementerades 5 metoder:

- openConnection: Skapar en koppling till den givna adressen, returnerar ”true” om en koppling skapades.
- closeConnection: Stänger nuvarande koppling.
- setQuestion: Ändrar typen av HTTP-frågan till nästa förfrågning, returnerar ”true” om ändringen godkändes.
- testConnection: Kontrollerar att en koppling finns och fungerar, returnerar ”true” om sådan finns.
- executeHTTPRequest: Skickar en HTTP-fråga med existerande koppling och returnerar svaret som en String.

För att få ovannämnda metoder att fungera skapades 5 variabler:

- Address: Ett String-objekt som håller reda på adressen till Elasticsearch.
- Mapping: Ett String-objekt som håller reda på vilken mapp i Elasticsearch informationen finns i.
- Connection: En java-klass som håller reda på den booleska förfrågan som skickas till Elasticsearch.
- URL: En java-klass som öppnar en anslutning med given adress.

- Open: En boolesk variabel som håller reda på om det finns en existerande anslutning eller inte.

Informationen för variablerna skrevs in i koden (se *figur 10*) utifrån information från Notifieds kontaktperson.

3.5.4 Steg 4 – Test av lösning

Uppgift: Få en testklass som imiterar Notified att skicka en boolesk förfrågan till API:et, som vidarebefordrar den till Elasticsearch. Sökträffarna ska returneras till Testklassen.

För att kontrollera att API:et fungerade från föregående steg så utvecklades en java-klass kallad ”LocalTestOfES_API” (se *figur 11* nedan) i LocalTest-paketet (nämnt i *3.5.3 Steg 3 - API*). Denna testklass skulle simulera Notified genom att skicka HTTP-förfrågningar.

Testningen började med exporteringen av API:et till ett annat paket (se avsnitt *3.5.3 Steg 3 – API*) som en JAR-fil. Detta för att simulera API:ets överföring till en annan miljö. Detta lyckades då testklassen beordrades att skapa en koppling med API:et via metoden *testConnection()*, vilket lyckades. Därefter skrevs en HTTP-förfrågan in manuellt och skickades till API:et. Eftersom historik från Elasticsearch returnerades kunde man bekräfta att förfrågan togs emot av API:et, vidarebefordrades till Elaticsearch samt att svaret returnerades till testklassen. Som en bekräftelse skrevs resultatet ut i konsolen.

ES_API.java | LocalTestOfES_API.java

```
1  package localTest;
2  import java.util.Scanner;
3  import api.ES_API;
4
5  public class LocalTestOfES_API {
6
7      public static void main(String[] args) {
8
9          Scanner scan = new Scanner(System.in);
10
11         System.out.println("Testing connection to database......");
12         System.out.println(ES_API.testConnection());
13
14         System.out.println("Testing fetch by search word: ");
15 //       String search = "/_search?";
16         String search2 = scan.next();
17         System.out.print(ES_API.executeHTTPRequest(search2));
18     }
19 }
```

Figur 11: Java-klassen "LocalTestOfES_API". Vid exekvering så körs metoden testConnection() i API:et och därefter skrivs en HTTP-förfrågan an användaren som vidarebefordras till Elasticsearch.

Med testningen avklarad så fanns nu en fungerande lösning som kunde hämta historik från Notifieds databas, lagra den hos en mellanhand som förstår booleska operationer i HTTP och vara tillgänglig för Notified.

3.5.5 Avslutande analys innan exportering

Med en fungerande lösning i testmiljön (se avsnitt *3.5.4 Test av lösning*) var nästa steg att exportera resultatet till Notifieds miljö. Men innan detta kunde göras kom en viktig fråga upp: Är API:et nödvändigt för att lösningen ska fungera? För att besvara denna fråga så undersöktes vilka ansvar som Elasticsearch, API:et och Notified hade. Resultatet blev följande:

Notified:

- Att skapa samt skicka HTTP-förfrågningar till API:et.
- Att ta emot resultatet. Resultatet ska vara i ett format Notified förstår.

API:

- Ta emot och vidarebefordra HTTP-förfrågningar från Notified till Elasticsearch.
- Ta emot och vidarebefordra resultat från Elasticsearch till Notified.
- Konvertera HTTP-förfrågningar och resultat till ett format som Elasticsearch och Notified förstår, om nödvändigt.

Elasticsearch:

- Ta emot HTTP-förfrågningar och tolka dessa.
- Returnera resultaten till API:et.

Vid uppställning av ansvar så kom det fram att API:et inte är nödvändigt av två anledningar: För det första kan Notified lagra adresser till vilka den skickar HTTP-förfrågningar till, vilket gör API:ets ansvar att vidarebefordra till rätt adress en onödig mellanhand. Dessutom är HTTP-formatet något som Elasticsearch förstår, vilket var den primära anledningen till att verktyget valdes (se avsnitt *2.3 Elasticsearch*).

För det andra så returnerar Elasticsearch resultatet i JSON, vilket är ett format som Notified förstår. Och om Notified kommunicerar direkt med REST-verktyget så kan den bara returnera resultatet direkt till avsändaren utan en mellanhand. Därmed blev API:et onödigt och kunde tas bort från lösningen.

3.6 Källkritik

Majoriteten av referenserna i avsnitt *8 Källförteckning* är officiella förstahandskällor hämtade från utvecklarens/företagets hemsida. Informationen hämtade från dessa källor är om verktygen de utvecklat och det är inte ovanligt att informationen kommer direkt från utvecklarna. Eftersom skaparna av verktygen anses vara de som vet mest om företagets produkter anses informationen vara trovärdig.

Undantagen till argumentet ovan presenteras och diskuteras nedan.

[1] är en Youtube-video där Todd Fredrich från Pearson eCollege går igenom principerna bakom REST. Pearson eCollege är en mjukvaruutvecklare av eLearning och ägs av Pearson PLC. Eftersom Todd Fredrich i videon har många års erfarenhet av REST så anses källan som trovärdig.

[2] Instructables.com är en hemsida där användare kan lägga upp projekt och redovisar hur man genomför dem. Projekten kan vara allting från matlagning till 3D-skrivning och är därmed inte begränsad till IT. Inga källreferenser eller information av publiceraren fanns tillgängliga, samt ingen information att projektet blivit granskat innan publicering. Dock så hade Eclipse stöd för utveckling av API och försök att efterlikna delar av projektet var framgångsrika. Slutsatsen blir därmed att projektet i fråga är trovärdig, men webbplatsen är inte tillförlitligt.

[5] Tutorialpoint.com är en websida med gratis onlineutbildningar, utvecklad av Tutorials Point (I) Pvt. Ltd. Deras handledningar är skrivna av proffs och läses igenom av ett team bestående av experter inom ämnesfrågor, tekniska författare, webbutvecklare och grafiska designers. Det är förbjudet att publicera deras arbete,

men det får användas i internt utbildningssyfte samt med hänvisningar till källan. Eftersom deras arbeten granskas så anses källan vara trovärdig.

[12] Denna referens var en workshop som skedde den 6 februari 2018 på Stora Varvsgatan 6a i Malmö. Workshoppen introducerade Elastics produkter, deras användningsområden och vad man ska tänka på vid användning. Man introducerades även till de nya möjligheterna som Elastic utvecklar och vad produkten X-Pack är för något. Workshoppen leddes av Eric Westberg, lösningsarkitekt på Elastic.
Eftersom Eric var anställd av Elastic samt hade arbetat med produkterna som han presenterade så anses han vara en trovärdig källa.

[15] *Kanban and Scrum – Making the Most of Both* är en bok som förklarar de agila arbetsmetoderna Scrum och Kanban. Eftersom boken används som undervisningsmaterial i LTH så anses den vara en trovärdig källa.

[19] Mattias Axelsson, Joel Guedj, Philip Adelgård mm är anställda på företaget Notified med flera års erfarenhet. Eftersom de är erfarna med Notified och dess utvecking anses deras kunskap om verktyget vara en trovärdig källa.

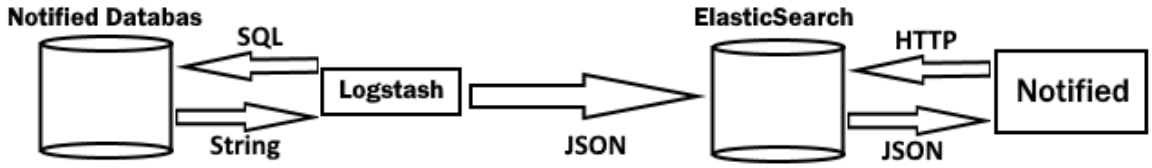
[20] NE.se har ämnesexperter som skriver texterna åt nationalencyklopedin. Dessa experter har i sin tur egna källor som referensverk och facklitteratur. Därmed anses NE.se vara en trovärdig källa.

[21] *A dictionary of the internet* är ett lexikon utgiven av Oxford Universitet. Eftersom boken är granskad så anses den vara en trovärdig källa.

4. Resultat

Detta kapitel innefattar en redogörelse av slutresultatet (se avsnitt 4.1 Slutresultat) samt exporten av lösningen från testmiljön till företagets miljö, d.v.s. Notifieds server (se avsnitt 4.2 Installationsprocessen)

4.1 Slutresultat



Figur 12: Slutresultatet som utvecklats under examensarbetet.

Slutresultatet består av fyra komponenter: Notfieds databas, Logstash, Elasticsearch och Notified, varav Logstash och Elasticsearch introducerades för företaget under examensarbetet. (se *figur 12* ovan).

Notifieds databas innehåller information om tidigare gjorda sökningar av Notifieds användare. Informationen finns uppdelad mellan flera datatabeller och är tillgänglig via SQL-förfrågningar.
Logstash är ansvarig för att skicka en SQL-förfrågan till databasen med kunskap om vilka datatabeller som är av intresse. Logstash startas manuellt, d.v.s. att Notifieds personal startar Logstash via kommandotolken. Logstash läser därefter sin konfigurationsfil där den förberedda SQL-frågan till Notifieds databas finns. Logstash levererar informationen i JSON-format till Elasticsearch.
Elasticsearch agerar som databas för Notified med informationen av intresse. Genom att spara data som Logstash förser den med kan Notified ställa ett obegränsat antal frågor och få snabb tillgång till informationen utan att behöva veta namnen på databastabellerna. Elasticsearch förstår booleska frågor i HTTP och returnerar resultaten i JSON, vilket är mycket bekvämt för Notified som använder dessa format.
Notified kommunicerar med användaren och skräddarsyr sökningar efter dess önskemål. Sökningar görs om till HTTP-förfrågningar som skickas ut till 3:e parters API:er och nu även till Elasticsearch, som har tillgång till tidigare gjorda sökningar som kanske inte längre finns i källans historik.

4.2 Installationsprocessen

Innan exporteringen av slutresultatet skedde hölls ett samtal med företagets kontaktperson. Eftersom säkerhet och integritet är områden som inte tagits hänsyn till under examensarbetet finns risken för att obehöriga får tillgång till information via Elasticsearch och/eller Logstash. Därmed beslutades det att lösningen inte skulle exporteras till Notifieds aktiva server, utan till en av Notifieds testservrar. Där skulle den säkert verifieras utan risk för företagets tjänster. Dessutom framkom det att Notifeds nya databas Amazon Aurora är av typen PostgreSQL. Detta innebär att om slutresultatet fungerar med Notifieds tidigare (postgreSQL) databas så ska det gå att koppla Logstash till Amazon Aurora.

Med tillgång till testservern kunde exporteringen påbörjas. Elasticsearch (med 3GB minnesutrymme) installerades. Det noterades att Elasticsearch måste startas manuellt för att vara tillgänglig, men då testservern alltid är påslagen ska informationen alltid vara tillgänglig. Därefter installerades Logstash. En ny mapp ”Notified” skapades bland Logstash filer där konfigurationsfilen ”logstash.conf” exporterades från testmiljön. Flera variabeländringar såsom lösenord och lokalisering av filer skrevs om för att överensstämma med den nya miljön. Kommentarer tillades i filen för att underlätta framtida ändringar.

Med båda programmen redo beordrades Logstash på att hämta all information från Notifieds (riktiga) databas till Elasticsearch. P.g.a. den enorma mängden information fick överföring pågå över natten. Postman installerades och med en HTTP-förfrågan kunde antalet rader i Elasticsearch jämföras med antalet rader i Notifieds databas. Då resultaten var lika så kunde man bekräfta att överföringen lyckats och därmed bevisat att Logstash kan både nå Notifieds databas samt leverera till Elasticsearch.

5. Slutsats

Detta kapitel handlar om hur slutresultatet påverkar samhället, en analys av hur arbetsmetoden påverkat examensarbetet samt att problemställningarna som ställts i avsnitt 1.6 besvaras.

5.1. Resultatets samhällspåverkan

Resultatet av examensarbetet ger Notified möjligheten att använda den sparade informationen från medier i framtida sökningar, även om informationen raderats från ursprungskällan. Därmed kommer företag och privatpersoner som använder Notified ha möjligheten att övervaka flera källor utan riskera att förlora information, även efter långa tidsperioder. Detta ger möjligheten att göra stora analyser. Företag med kampanjer eller projekt som pågår i månader (eller t.o.m. år) är intresserade av hur medierna kommenterar deras utveckling samt hur deras ståndpunkt ändras med tiden. Denna information hjälper företag att planera sitt nästa steg i marknadsföring och t.o.m. ta kontakt med en eller flera källor som levererar konstruktiv kritik av intresse. Sammanfattningsvis minskar examensarbetet risken för att användare av Notified gör felaktiga analyser p.g.a. information som inte inkluderats då källans hantering av historik lett till att information försvunnit.

Dock finns en risk för att Notified sparar information som tagits bort p.g.a. felaktiga påståenden, konfidentiell information som publicerats av misstag eller annan missledande historik. En källa har rätt att ändra informationen den presenterar. Om Notified sparar alla ändringar som gjorts kan detta ge en felaktig bild av källan, om missledande information lever kvar och fortsätter att presenteras. Dessutom ges användarna felaktig information. Eftersom historiken i Notifieds databas är oberoende av ursprungskällan så måste källan kontakta företaget för att få informationen ändrad eller raderad. Detta är något som källor lätt kan missa då detta är en tjänst just för företagets kunder och källor är möjligtvis inte medvetna om företaget eller Notifieds tjänster.

5.2 Analys av arbetsmetoden

Som presenterat i avsnitt *3.2 Arbetsmetod* gick arbetsmetoden ut på att iterera mellan utveckling och analys inom varje fas av examensarbetet (se avsnitt *3.1 Examensarbetets faser*). Detta gjorde att flera analyser kunde göras genom hela examensarbetet och tillät flexibilitet om ändringar och lösningsalternativ. Det största exemplet är valet att ta bort API:et från lösningen, något som gjordes därför att det visade sig vara onödigt, även om den redan utvecklats samt att ändringen kom väldigt sent. Dessutom såg arbetsmetoden till att en uppgift utvecklades i taget och att inget lämnades halvfärdigt. Slutsatsen är därmed att arbetsmetoden var väl anpassad för examensarbetet.

5.3 Besvarande av problemformuleringar

Nedan besvaras de problemställningar som ställdes i 1.6. Svaren på dessa problem kan hittas i kapitel 3 Analys & Metod.

Hur får man gamla sökresultat att dyka upp i nya, unika sökningar om nyckelorden inte är identiska men ämnet är relevant?

I ett samtal med företagets kontaktperson (se avsnitt *3.3 Förstudier*) diskuterades det om hur detta problem kunde lösas. Svaret blev att målet med examensarbetet är att ge Notified tillgång till all resultat från tidigare sökningar, oavsett vilka sökningar som sker. Om en sökning efter specifik information görs av en användare så är det Notified ansvar att skicka dessa sökord till alla 3:e parter, där parterna bestämmer vilken information som är relevant att returnera. Detta är därmed inte ett problem som inte behöver lösas av en koppling mellan två punkter.

Hur undviks dubblering av resultat?

I ett samtal med företagets kontaktperson (se avsnitt *3.3 Förstudier*) diskuterades det om hur detta problem kunde lösas. Svaret blev att eftersom Notified skickar ut sin förfrågan till 3:e parter oberoende av vad varandras svar är så är detta inget som lösningen behöver ta hänsyn till. Dessutom ska all information från tidigare sökningar vara tillgänglig, oavsett hur stor eller liten del används i slutändan. Däremot är detta en möjlig vidareutveckling, då flera identiska resultat kan försämra användarvänligheten om flera sökresultat hänvisar till samma information.

Kan man länka Amazon Aurora-servern till ”wrappen”?

Detta var en fråga som uppkom när det första lösningsförslaget med API:et blev till (se avsnitt *3.3 Förstudier*). Dock eftersom inga lämpliga wrappers fanns tillgängliga (se kapitel *3.4 Analysfasen*) så ersattes idén om en wrapper med en mellanhand istället och därmed slutade frågan vara relevant. Men för att besvara frågan ”Kan man länka Amazon Aurora-servern med lösningen” så är svaret ja då Amazon Aurora är av typen PostgreSQL, vilket är samma typ av databas som (framgångsrikt) arbetats med i testmiljön och vid exporteringen av lösningen.

Är ett API nödvändigt för en fungerande lösning?

Som besvarat i avsnitt *3.5.5 Avslutande analys innan exportering* så ansågs API:et inte vara nödvändigt för att lösningen skulle fungera. API:et som utvecklats under examensarbetet fungerade och kunde ha ingått i slutlösningen. Dock fanns risk för att API:et inte kunde hantera meddelanden korrekt mellan Notified och Elasticsearch. Exempel på detta är Eclipse begränsning av speciella tecken (se avsnitt *2.5 Eclipse*). Detta samt att alla arbetsuppgifter hade tagits av antingen Elasticsearch eller av Notified gjorde att API:et hade ingen riktig arbetsuppgift och därmed onödig.

6. Framtida utvecklingsmöjligheter

Detta kapitel presenterar flera möjliga utvecklingsområden som fortsätter utvecklandet av examensarbetet.

Även om Notified har fått en ny möjlighet i sin bevakningstjänst så finns utrymme för vidareutveckling av examensarbetet.

6.1 Säkerhet & Automatisering

Då examensarbetet inte tagit hänsyn till datasäkerhet eller skydd för intrång så kan en vidareutveckling av säkerheten vara nödvändig innan Notified lanserar den till sin live-version. Dessutom så måste både Logstash (som levererar information) och Elasticsearch (som sparar vad Logstash förser den med) startas av användaren. Detta kan ersättas av ett program eller mjukvara som ansvarar för att Logstash frekvent hämtar ny information samt att Elasticsearch alltid är tillgänglig.

6.2 Exportering till Notified

Som tidigare nämnt så är Amazon Aurora av PostgreSQL och Elasticsearch är därmed redo att användas av Notified. Dock så finns flera utmaningar med nya källan, såsom presentation av information. På grund av upphovsrättslagen så får hela artiklar ej sparas, utan endast utdrag får göras. Därmed så presenteras informationen annorlunda jämfört med Notifieds andra källor, t.ex. Twitter och Facebook.

Dessutom så är chansen stor att databasen presenterar information som andra källor redan har och därmed ger Notified en dubbel sökträff för varje resultat. Detta skulle gå emot målet med examensarbetet, att presentera resultat som inte andra källor gett, och dessutom inte vara användarvänligt.

7. Ordförklaringar

3:e parter: Källor, företag och personer som inte har direkta kopplingar till företaget. Twitter, Facebook och Youtube är exempel på 3:e parter som Notified hämtar information från men där källorna inte nödvändigtvis känner till Notified.

Användare: Företagets kunder, konsumenter av Notified.

Cykel: (En) Iteration. Flera cyklar innebär att processen itereras flera gånger, m.a.o. att processen repeteras.

HTTP: Hypertext Transfer Protocol.

Eclipse: En utvecklingsmiljö för olika programspråk. Se avsnitt 2.5 i *Teknisk Bakgrund*.

Elasticsearch: Ett mjukvaruverktyg som kan agera sökmotor och/eller databas. Se avsnitt 2.3 i *Teknisk Bakgrund*.

Företaget: Mjukvaruföretaget Notified. Grundades 2010 och har idag kontor i både Malmö och Stockholm. Arbetsgivaren av examensarbetet. Se avsnitt 1.1. Får ej förväxlas med verktyget Notified.

Logstash: En mjukvara som hämtar och transporterar information. Se avsnitt 2.4 i *Teknisk Bakgrund*.

Notified: Företagets bevakningsverktyg för sociala medier. Får ej förväxlas med företaget Notified.

Notifieds kontaktperson: Mattias Axelsson.

Notifieds anställda: Ett flertal personer anställda av företaget Notified. De med mest inflytande av arbetet var Mattias Axelsson, Philip Adelgård och Joel Guedj.

PgAdmin4: Utvecklingsplatformen för PostgreSQL. Se avsnitt 2.7 i *Teknisk Bakgrund*.

PostgreSQL: Ett databassystem. Se avsnitt 2.7 i *Teknisk Bakgrund*.

Postman: En utvecklingsmiljö för HTTP och API:er. Se avsnitt 2.6 i *Teknisk Bakgrund*.

REST: Ett designkoncept för hantering och strukturering av information. Se avsnitt 2.2 i *Teknisk Bakgrund*.

Sociala medier: Sociala nätverk som låter personer att kommunicera genom t.ex. text, bild och/eller ljud. Exempel är Facebook, YouTube och Twitter. [20]

URI: Uniform Resource Identifier.

Wrapper: Se avsnitt 2.1 i *Teknisk Bakgrund*.

8. Källförteckning

[1] *Intro to REST (aka. What Is REST Anyway?)* [2018-02-19] HTTPs://www.youtube.com/watch?v=llpr5924N7E

[2] *How To Develop And Use Java API In Eclipse* [2018-01-16] HTTP://www.instructables.com/id/How-to-Develop-and-Use-a-Java-API-in-Eclipse/

[3] *Elasticsearch Reference [6.1] – Search APIs* [2018-02-28] HTTPs://www.elastic.co/guide/en/elasticsearch/reference/6.1/search.html

[4] *PostgreSQL JDBC Driver* [2018-01-17] HTTPs://jdbc.postgresql.org/download.html

[5] *PostgreSQL - JAVA Interface* [2018-01-17] HTTPs://www.tutorialspoint.com/postgresql/postgresql_java.htm

[6] *About the Eclipse Foundation* [2018-02-15] HTTPs://www.eclipse.org/org/

[7] *Eclipse desktop & web IDEs* [2018-02-15] HTTPs://www.eclipse.org/ide/

[8] *Postman* [2018-02-15] HTTPs://www.getpostman.com/postman

[9] *Elasticsearch: RESTful, Distributed Search & Analytics* [2018-02-22] HTTPs://www.elastic.co/products/elasticsearch

[10] *Logstash: Collect, Parse, Transform Logs* [2018-02-22] HTTPs://www.elastic.co/products/logstash

[11] *pgAdmin – PostgreSQL Tools* [2018-02-23] HTTPs://www.pgadmin.org/

[12] *Foocafé - Elasticsearch best practices for performance* [2018-02-06] HTTP://www.foocafe.se/malmoe/events/1738-elasticsearch-best-practises-for-performance-and-scale

[13] *PostgreSQL - What is Postgres?* [2018-02-20] HTTPs://www.postgresql.org/docs/6.3/static/c0101.htm

[14] *Notified – Marknadens bästa lösning för mediebevakning och kommunikation* [2018-01-08] HTTPs://www.notified.com

[15] Kniberg, Henrik och Skarin, Mattias. *Kanban and Scrum – Making the Most of Both.* Kristianstad: C4Media, cop. 2009. ISBN: 978-0-557-13832-6.

[16] *Integrate Postgresql and Elasticsearch | ZomboDB* [2018-01-30] HTTPs://www.zombodb.com/

[17] *PostgreSQL Elastic Search foreign data wrapper* [2018-01-30] HTTPs://github.com/matthewfranglen/postgres-elasticsearch-fdw

[18] *JDBC Input plugin* [2018-01-30] HTTPs://www.elastic.co/guide/en/logstash/current/plugins-inputs-jdbc.html

[19] Anställd personal hos Notified: Mattias Axelsson (kontaktperson), Joel Guedj, Philip Adelgård, med flera.

[20] *Sociala medier – Uppslagsverk* [2018-05-19] HTTPs://www.ne.se/uppslagsverk/encyklopedi/l%C3%A5ng/sociala-medier

[21] Darrel Ince. *A Dictionary of the Internet (3 ed.).* Oxford: Oxford University Press. 2013. eISBN: 9780191744150.

9. Appendix

9.1 Frågor inför möte med företaget Notified.

1. Vilka resurser tros behövas eller användas under examensarbetet?
2. Vilka avgränsningar och/eller format ska slutprodukten vara i?
3. Vad ska jag undvika att ändra i under examensarbetet?
4. Hur ”nytt” är detta arbete för Notified, världsunikt?
5. Har ni några krav på slutprodukten?
6. Hur kommer jag att arbeta med databasen?

9.2 Krav från företaget Notified angående lösning.

1. Examensarbetet involverar enbart lösningar för operativsystemet Windows.
2. API:et som söker igenom Notifieds databas ska vara ett booleskt sök-API.
3. Nuvarande datastrukturen i Notifieds PostgreSQL-databas ska behållas, dock kan tillägg i form av datatabeller förekomma.