Toni Kujanpää

Hakualustan valitseminen ja päivittäminen digitaalisen sisällön kauppaan

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

4.5.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Toni Kujanpää Hakualustan valitseminen ja päivittäminen digitaalisen sisällön kauppaan 46 sivua 4.5.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Ohjelmistotuotanto
Ohjaajat	Lehtori Simo Silander Digitalist Group Service Manager Sami Viitaniemi

Hakualustat ovat olennainen osa nykyajan verkkosovelluksia. Niiden avulla käyttäjät kykenevät hakemaan tallennettua dataa erittäin nopeasti isoistakin datamääristä. Hakualustoja hyödynnetään varsinkin verkkokaupoissa ja videopalveluissa.

Insinöörityön tavoitteena oli päivittää työn tilanneen yrityksen käytössä oleva Apache Solr -hakualusta uusimpaan versioon, sekä selvittää, mitä muita hakualustoja on, vertailla hakualustoja keskenään ja lopuksi esittää yritykselle toimintaehdotus hakualustan mahdollisesta vaihtamisesta ja antaa ohjeita hakualustan valitsemiseen. Yritys oli tyytyväinen käytössään olevaan Apache Solriin, mutta halusi kuitenkin selvitettävän vaihtoehtoja sille tulevia projekteja varten.

Työ aloitettiin perehtymällä ensin Apache Solriin ja sen toimintatapaan. Piti myös selvittää yrityksen sovelluksessa käytettyjen muiden työkalujen, palveluiden ja ohjelmointikielen perusteet ennen kuin varsinaista päivittämistä voitiin aloittaa. Solrin päivitysprosessi alkoi testiympäristön luomisesta, jossa luotiin paikallisesti omalle tietokoneelle ympäristö, jossa muutoksia voitiin testata. Tämän jälkeen voitiin siirtyä virheenkorjaukseen ja ohjelmointiin. Solrin päivittämisen jälkeen voitiin siirtyä työn toiseen päätehtävään eli hakualustojen vertailuun. Vertailu suoritettiin ottamalla ensin selvää, mitä hakualustoja on olemassa, valitsemalla niistä olennaiset vaihtoehdot, selvittämällä niiden ominaisuudet ja lopuksi vertailemalla niitä keskenään.

Insinöörityön tuloksena oli Apache Solr -hakualusta päivitettynä uusimpaan versioon yrityksen sovelluksessa sekä hakualustojen vertailun tulokset. Vertailun tuloksien perusteella yritykselle annettiin toimintaehdotus Solrin mahdollisesta vaihtamisesta ja mitä yrityksen tulee ottaa huomioon tulevissa projekteissa hakualustaa valittaessa. Toimintaehdotuksessa esitettiin, ettei yritys vaihda käytössä olevaa Apache Solria, sillä se oli toistaiseksi riittävä ja vaihtoprosessi olisi mahdollisiin hyötyihin nähden liian vaativa. Vertailu tuotti yritykselle arvokasta materiaalia tulevia projekteja varten, joissa tulee valita sovellukselle hakualusta.

Avainsanat	Apache Solr, Scala, Tomcat, hakualusta, hakukone

Author	Toni Kujanpää
Title	Selecting and Updating a Search Platform for Digital Content Store
Number of Pages	46 pages
Date	4 May 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Simo Silander, Senior Lecturer Sami Viitaniemi, Service Manager, Digitalist Group

Search platforms are an essential part of modern web applications. They enable users to search stored data very quickly even from large amounts of data. Search platforms are used especially in online shops and video services.

The objective of the thesis was to update Apache Solr search platform used by the company to the latest version, find out what other search platforms there are, compare the search platforms, and finally present a proposal for a potential change of the search platform and provide guidelines for selecting a search platform. The company was happy with the Apache Solr in use but wanted to find out about possible alternatives for its future projects.

The work was started by first getting to know Apache Solr and the way it works. It was also needed to study the basics of other tools, services, and programming language used in the company's application before starting the actual updating. The update process began with the creation of a test environment which was a local environment where changes could be tested. After that, it was possible to start debugging and programming. After updating Solr, it was possible to move to the second main task of the work, which was to compare the search platforms. The comparison was made by first identifying what other search platforms exist, selecting the relevant alternatives for the comparison, finding out their features, and then comparing them with each other.

The result of the thesis was the Apache Solr search platform implementation updated to the latest version in the company's application and the results of the search platform comparison. Based on the results of the comparison, the company was given an operational proposal for the possible replacement of Apache Solr and what the company should take into account in future projects when selecting a search platform. The action proposal suggested that the company should not replace the existing Apache Solr because it was sufficient enough for the time being and the replacement process would be too demanding in relation to the potential benefits. The comparison provided the company with valuable material for future projects that require a search platform for the application.

Keywords	Apache Solr, Scala, Tomcat, search platform, search engine

Sisällys

Lyhenteet

1 Johdanto 1

2 Hakualustat 2

3 Lähtökohta 4

3.1 Digitalist Digital Content Store -projekti 5
3.2 Scala-ohjelmointikieli ja Scalatra-sovelluskehys 7
3.3 PostgreSQL-tietokantajärjestelmä 9
3.4 Apache Tomcat -verkkopalvelin 10
3.5 Apache Solr -hakualusta 12

4 Apache Solr -hakualustan toiminta 14

4.1 Indeksointi ja datan tallentaminen 15
4.2 Datan hakeminen 16
4.3 Solr-indeksin replikointi 18

5 Apache Solrin päivittäminen 20

5.1 Koodin kääntäminen 21
5.2 Testiympäristön luominen 22
5.3 Virheenkorjaus ja ohjelmointi 23

6 Hakualustojen vertailu 25

6.1 Elasticsearch 26
6.2 Splunk 28
6.3 Sphinx 30
6.4 Algolia 32
6.5 Amazon CloudSearch 34

7 Vertailun yhteenveto ja toimenpide-ehdotukset 36

7.1 Vertailun yhteenveto 36
7.2 Toimenpide-ehdotukset 41

8 Yhteenveto 42

Lyhenteet

API	Application programming interface eli ohjelmointirajapinta on kokoelma valmiita funktioita ja toimintoja, joiden avulla voidaan luoda sovelluksia, jotka käyttävät toisen palvelun ominaisuuksia tai dataa.
HTTP	Hypertext Transfer Protocol on protokolla, jota verkkopalvelimet ja selaimet käyttävät tiedonsiirrossa.
DCS	Digital Content Store on nimi projektille, jossa työ tehtiin.
DES	Digitalist Experience Store on käytännön toteutus Digitalist Digital Content Storesta, jota käytetään erään autovalmistajan pääyksiköihin toteutetusta sovelluskaupasta.
JVM	Java Virtual Machine eli Java-virtuaalikone, joka mahdollistaa Java-ohjelmien ajamisen laitteessa.
NoSQL	NoSQL (non SQL tai Not only SQL) on tietokantaratkaisu, jonka toimintaperiaate ei perustu relaatiotietokantamalliin, mutta voi tukea myös joitain relaatiotietokannoille tyypillisiä SQL-tyylisiä hakukieliä.
REST	Representational State Transfer (REST) on HTTP-protokollaan perustuva arkkitehtuurimalli, jonka avulla toteutetaan ohjelmointirajapintoja.
SaaS	Software as a Service (SaaS) tarkoittaa ohjelmiston hankkimista verkkopalveluna perinteisen lisenssipohjaisen tavan ja ohjelmiston asentamisen sijaan.

1 Johdanto

Hakualustojen tehtävä on tarjota turvallinen menetelmä tallentaa, järjestää ja ennen kaikkea noutaa dataa mahdollisimman nopeasti. Niitä käytetään esimerkiksi verkkokaupoissa tuotteiden etsimisessä, sillä tuotteita on yleensä niin paljon, että olisi vaikeaa löytää etsimänsä ilman hakuominaisuutta. Hakualustat ovat tärkeä osa nykyajan isoja pilvipalveluita ja verkkosovelluksia, joissa dataa on paljon, mutta sitä pitää myös pystyä käsittelemään nopeasti ja vikaturvallisesti. Nopea datan haku luo sulavan käyttökokemuksen, johon käyttäjät ovat tottuneet. Datan määrä kasvaa jatkuvasti ja siksi juurikin hakujen nopeus korostuu hakualustojen välisissä vertailuissa.

Insinöörityössä perehdytään erityisesti Apache Solr -hakualustaan ja sen toiminnallisuuksiin sekä mahdollisuuksiin. Tavoitteena on vertailla Solria muihin vastaaviin hakualustoihin ja selvittää, mikä niistä olisi sopivin työn tilanneen yrityksen tarkoitukseen. Solria verrataan muutamaan yleisimpään hakualustaan ja selvitetään niiden vahvuudet, heikkoudet ja merkittävimmät eroavaisuudet. Hakualustojen vertailun lisäksi insinöörityön tavoitteena on myös päivittää käytössä oleva Solr uusimpaan versioon.

Työ tehdään Digitalist Group -yritykselle. Digitalist Group omistaa Digitalist Digital Content Store -palvelun (DCS), joka hyödyntää Solr-hakualustaa ja sen ominaisuuksia digitaalisen sisällön kaupassaan. Työssä käydään ensin läpi projektin lähtökohta ja arkkitehtuuri sekä Apache Solr -hakualustan toiminta, jonka jälkeen perehdytään Solrin päivittämiseen uusimpaan versioon ja hakualustojen vertailuun. Lopuksi työssä käsitellään työn johtopäätökset ja toimenpide-ehdotukset yritykselle sekä lyhyt yhteenveto työstä.

Työn tuloksena on Digitalist Digital Content Storeen implementoidun Solrin toteutus uusimman version mukaisesti. Hakualustojen vertailun tulosten perusteella insinöörityössä päädytään myös lopputulemaan siitä, tulisiko yrityksen vaihtaa hakualustaa, mitä hyötyjä ja haittoja vaihtamisesta olisi ja mikä hakualusta sopisi parhaiten kyseiseen tarkoitukseen. Vaikka tuloksena olisi jatkaa samalla hakualustalla, yritys saa arvokasta tietoa tulevia projekteja varten, joissa joudutaan valitsemaan hakualustojen väliltä sopivin vaihtoehto.

2 Hakualustat

Hakualusta tai hakukone on ohjelmisto, joka hakee syötetyillä avainsanoilla asiakirjoja ja tiedostoja ja palauttaa tuloksena dataa, joka sisältää näitä avainsanoja. Hakukone-sanasta monelle tulee ensimmäisenä mieleen Google. Google on kuitenkin verkkohakukone, joka on suunniteltu etsimään tietoa World Wide Webistä. Tässä työssä käsiteltävät hakukoneet on suunniteltu yritysten käyttöön tietokannoista ja indekseistä hakemiseen, eivätkä ne ole sama asia kuin verkkohakukoneet. Tässä työssä käsiteltäviä hakukoneita voidaan kutsua myös hakualustoiksi ja kutsunkin niitä tässä työssä hakualustoiksi välttääkseni niiden sekoittamisen verkkohakukoneisiin. Tällaisia hakukoneita kutsutaan myös yrityshakukoneiksi (enterprise search engine).

Hakukoneet käsittelevät dataa, joka on niin sanotusti jäsentymätöntä (unstructured). Voidaan ajatella, että jäsentymättömyydellä tarkoitetaan tietokoneen näkökulmaa tekstiin, sillä tietokone näkee tekstin vain virtana merkkejä. Merkkivirta on jäsennettävä käyttämällä kielikohtaisia sääntöjä, joiden avulla rakennetta voidaan purkaa ja tehdä se haettavaksi. Tämä on juurikin se, mitä kaikki hakukoneet tekevät. [1.]

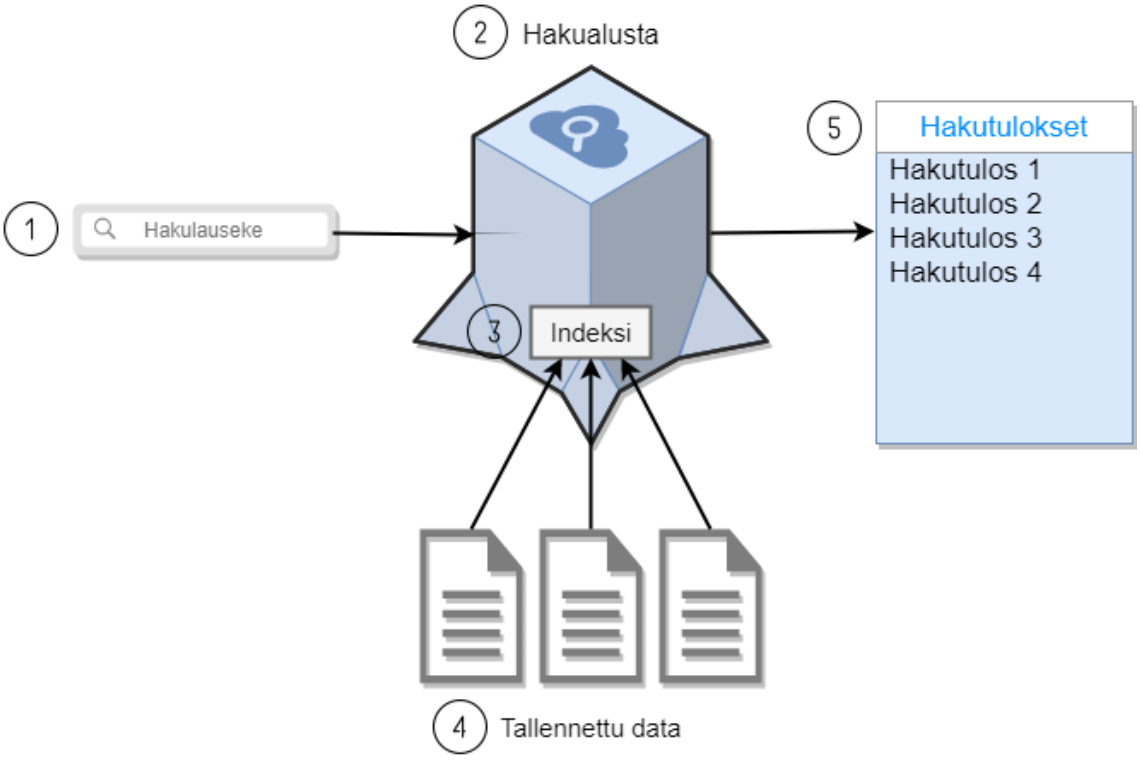
Jokainen meistä on joskus käyttänyt jotain hakualustaa etsiessään tietoa verkkosivuilla tai -sovelluksissa. Hyvä esimerkki on vaikkapa Youtube, jossa käyttäjät etsivät videoita hakusanoillaan. Youtuben hakuominaisuus perustuu hakualustaan, joka on tallentanut dataa kaikista Youtubeen lisätyistä videoista ja lajitellut ne niin, että se pystyy nopeasti noutamaan sieltä haluttua dataa, eli tässä tapauksessa videoita, jotka liittyvät hakusanoihin. Hakualustoissa on myös yleensä jonkinlainen systeemi, joka määrittelee, mikä hakutuloksista on merkityksellisin ja järjestää ne sen mukaan. Merkityksellisyyden perusteet voivat vaihdella, ja ohjelmoija pystyy vaikuttamaan niihin. Esimerkiksi Youtube todennäköisesti painottaa eniten tuloksia, joissa avainsana esiintyy videon otsikossa tai videon ladanneen käyttäjän nimessä. Solr-hakualustassa on hyvä sisäänrakennettu merkityksellisyyden arviointi, joka on yleensä riittävä ilman, että sitä tarvitsee erikseen muuttaa.

Hakualustoja on useita, ja ne toimivat hieman eri tavoin, mutta kaikilla niillä on sama tavoite: hakea dataa mahdollisimman nopeasti isoista tietokannoista. Osa isoista yrityksistä on myös luonut itse hakualustoja omaan käyttöönsä. Yleisimpiä hakualustoja ovat muun muassa Apache Solr, Elasticsearch, Splunk, Sphinx, Algolia ja Amazon CloudSearch, joihin tutustumme työssä myöhemmin.

Hakualustan toimintaperiaate koostuu viidestä osasta:

1. Hakulauseke, jolla käyttäjä suorittaa kyselyn eli hakee tuloksia.
2. Hakualusta, joka sisältää algoritmit, jotka jäsentävät kyselyn ja määrittelevät merkityksellisyydet.
3. Indeksi eli hakemisto, joka sisältää jokaisen tallennetun dokumentin jokaisen sanan sijainnin ja pystyy siten hakemaan hakusanalla haettuja dokumentteja erittäin nopeasti.
4. Tallennettu data eli haettavat dokumentit.
5. Hakutulokset, jotka hakualusta tuottaa hakulausekkeen perusteella.

Nämä viisi osaa ja hakuprosessi on havainnollistettu kuvassa 1. Hakuprosessi alkaa käyttäjän kirjoittaessa hakulausekkeensa ja lähettäessään sen eteen päin. Hakulauseke ohjautuu hakualustalle, joka jäsentää sen ja etsii indeksin ja merkityksellisyysalgoritmien avulla parhaat hakutulokset tallennetusta datasta. Indeksi on hakualustan sisäänrakennettu osa. Hakualusta palauttaa lopuksi loppukäyttäjälle merkityksellisimmät hakutulokset.



Kuva 1. Hakualustojen toimintaperiaate.

Merkityksellisyydellä tarkoitetaan sitä, että hakutulokset vastaavat sitä, mitä käyttäjä on yrittänyt hakea. Kun hakutulokset on järjestetty merkityksellisyyden mukaan,

ensimmäisenä tulisi olla tulokset, jotka ovat mahdollisesti lähimpänä sitä, mitä käyttäjä haullaan etsii. Esimerkiksi jos käyttäjä hakee verkkokaupasta Adidaksen kenkiä, tulisi hakutuloksista ensimmäisenä olla Adidaksen kengät ja vasta sen jälkeen muiden merkkien kenkiä. Jos verkkokaupassa ei ole Adidaksen kenkiä, niin hakutulokset ovat kuitenkin järjestyksessä merkityksellisyyden mukaan. Tällöin merkityksellisyyteen voi vaikuttaa esimerkiksi, mitä kenkiä Adidaksen kenkiä hakeneet käyttäjät ovat eniten katsoneet. Esimerkiksi Apache Solr mahdollistaa merkityksellisyyden arvioinnin siten, että eri ominaisuuksille voidaan antaa painoarvoja, joita vertailemalla hakutuloksissa Solr päättelee merkityksellisyysjärjestyksen.

3 Lähtökohta

Ennen kuin varsinaista työtä aloitettiin, täytyi selvittää lähtökohta ja arkkitehtuuri tarkemmin. Digitalist Group -yrityksen tuote oli DCS (Digitalist Digital Content Store), joka oli jo täysin toimiva kokonaisuus. DCS on palvelinpuolen (back end) toteutus digitaalisen sisällön kaupoille. Sen päälle voidaan luoda useita erilaisia verkko- ja sovelluskauppatoteutuksia eri tarkoituksiin ja sitä pystytään muokkaamaan eri asiakkaiden tarpeiden mukaan heille sopivaksi. Yksi näistä toteutuksista on DES (Digitalist Experience Store), joka oli työn aloitushetkellä ainut tuotannossa oleva DCS-toteutus.

Ongelmana oli, että DCS-projektissa käytössä ollutta Apache Solr -hakualustaa ei ollut päivitetty uusimpaan versioon vuosiin. Projektissa käytetty versio oli 3.4. Uusin Solr-versio oli työn aloitushetkellä 7.3, joten projektissa käytetty Solr oli paljon jäljessä uusimmasta versiosta. Projektissa oli myös yritetty toteuttaa Embedded Solr -ominaisuutta, mutta sitä ei koskaan ole saatu täysin toimimaan. Embedded Solria ei enää suositella käytettäväksi eikä sille ollut varsinaista tarvettakaan, joten tehtävänäni oli myös poistaa kaikki Embedded Solriin liittyvät koodit ja siten puhdistaa koodia.

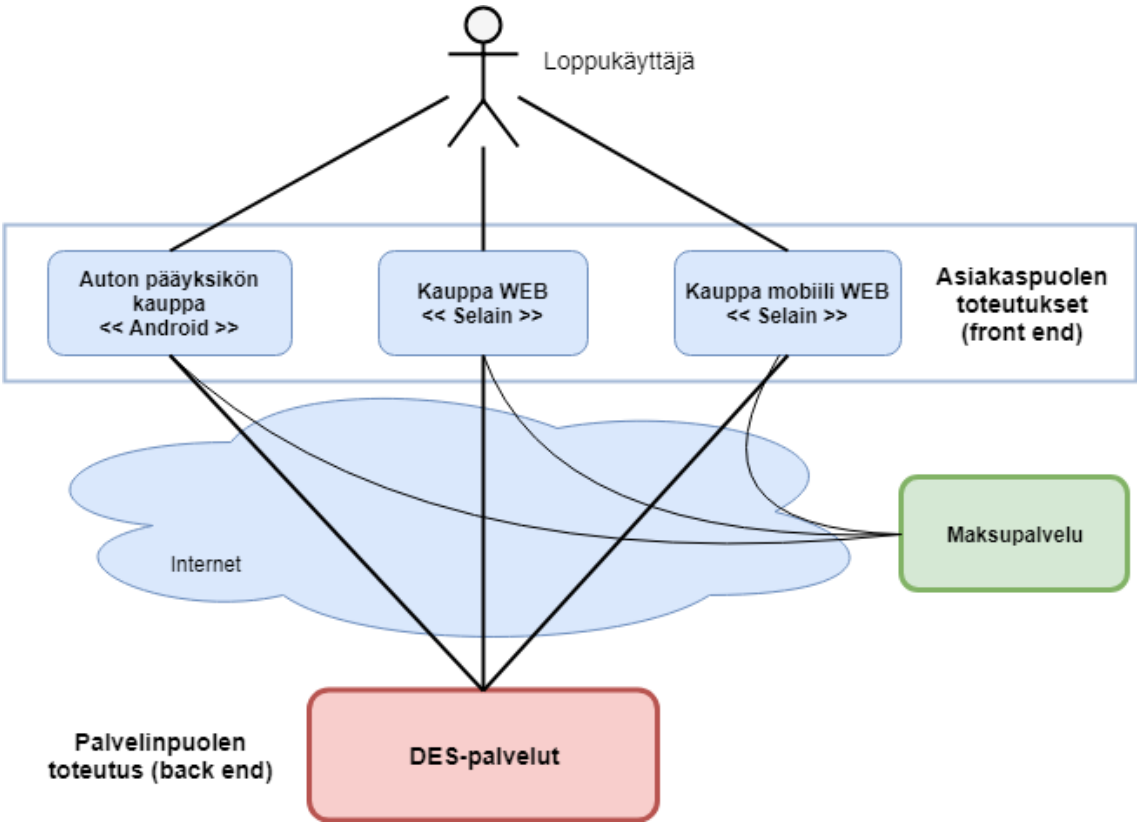
Solria käytetään projektin Discovery- ja Publishing-komponenteissa tuoteindeksien luomisessa, joka mahdollistaa tuotteiden, kategorioiden ja laitteiden listaamisen sekä etsimisen tuotteittain nimikkeen, kuvauksen, tekijän jne. mukaan. Perehdymme työssä tuoteindeksien luomiseen myöhemmin Solrin yhteydessä. Toimiakseen Solr tarvitsi ennen versiota 5.0 Java Servlet Container -kehyksen eli verkkopalvelimen. Solrin

mukana tulee Java Servlet Container -kehys nimeltä Jetty, mutta kyseisessä projektissa on käytössä vastaava kehys nimeltä Tomcat.

Itse palvelinpuolen koodi on ohjelmoitu Scala-ohjelmointikielellä. Scala on tehty Javan päälle, joten sen lähdekoodi ajetaan lopulta Java-virtuaalikoneella (Java Virtual Machine). Scala on valittu projektiin palvelinpuolen ohjelmointikieleksi, koska koko arkkitehtuuri on rakennettu Javan ympärille. Solr, Tomcat ja Scala ovat kaikki Java-pohjaisia toteutuksia. Scala on valittu palvelinpuolen (back end) ohjelmointikieleksi myös sen funktionaalisten ominaisuuksien vuoksi. HTTP-pyynnöt ja REST-arkkitehtuurin hoitaa Scalatra-sovelluskehys. HTTP (Hypertext Transfer Protocol) on protokolla, jota verkkopalvelimet ja selaimet käyttävät tiedonsiirrossa. REST (Representational State Transfer) on HTTP-protokollaan perustuva arkkitehtuurimalli, jonka avulla toteutetaan ohjelmointirajapintoja. Scala on hyvä XML-tyyppisen datan käsittelyssä ja siksi se sopii hyvin DCS:n kaltaiseen verkkosovellukseen, jossa käsitellään paljon dataa. Dataa voisi käytännössä käsitellä myös JSON-muodossa, mutta DCS-projektissa lähes kaikkea dataa käsitellään XML-muodossa, sillä Scala on erikoistunut juurikin sen käsittelyyn.

3.1 Digitalist Digital Content Store -projekti

Digitalist Digital Content Store on Digitalist Groupin omistama digitaalisen sisällön kauppa. Sitä käytettiin työn aloitushetkellä erään autonvalmistajan autoissa olevissa pääyksiköissä, joissa tuote kulkee nimellä Digitalist Experience Store (DES). DES on siis yksi toteutus DCS:n päälle. DES on sovelluskauppa, josta auton omistaja voi ostaa ja ladata hyödyllisiä sovelluksia autoonsa. Sovelluskauppaan pääsee auton pääyksikön kautta sekä älypuhelimilla tai millä tahansa laitteella, jossa on verkkoselain. Osa sovelluksista on tarkoitettu ladattavaksi älypuhelimeen ja osa auton pääyksikköön. Sovelluksista osa on maksullisia ja sitä varten DCS:ssä on toteutettu oma komponentti maksamista varten. Kuva 2 kuvaa DES:n toimintaympäristöä, jossa loppukäyttäjä käyttää DES-sovelluskauppaa joko älypuhelimen tai tietokoneen verkkoselaimella tai auton pääyksikön Android-pohjaisella käyttöliittymällä. Kaupan sisältö eli DES:n tapauksessa sovellukset saadaan näkyviin vain internetin välityksellä, sillä niiden fyysinen sijainti on palvelimilla.



Kuva 2. Digitalist Experience Store -sovelluskaupan toimintaympäristö

DES on siis vain yksi Digitalist Digital Content Storen päälle rakennettu toteutus, mutta koska se oli työn aikana ainut tuotannossa oleva toteutus, se määritteli, mitä ominaisuuksia DCS:ään tuli toteuttaa ja kuinka sen tuli toimia.

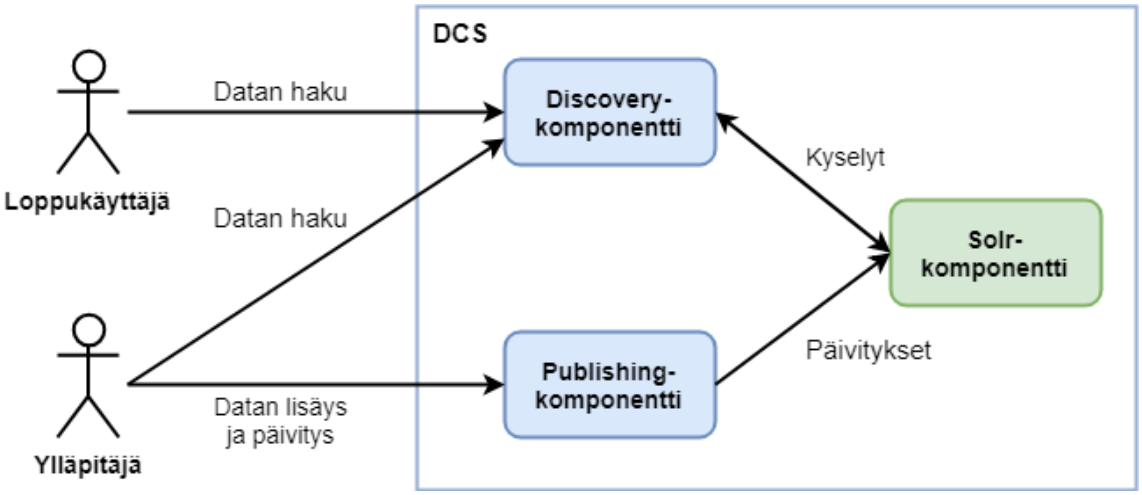
DCS sisältää useita komponentteja, jotka toimivat keskenään mahdollistaakseen toimivan kokonaisuuden. Tämän työn kannalta tärkeimmät komponentit ovat Publishing (julkaisu) ja Discovery (löytäminen).

Publishing on komponentti, jonka kautta kauppaan lisätään sisältöä, eli DES:n tapauksessa sovelluksia. Publishing tekee järjestelmästä skaalautuvan ja mukautuvan. Se hakee dataa järjestelmän sisäisistä komponenteista, muuntaa ne yleiseen formaattiin asiakaspuolta (front end) varten ja toimittaa sisällön asiakaspuolelle indeksoidun tietokannan, Solrin kautta. Solr hallinnoi datan replikoinnin fyysisiin asiakaspuolen laitteisiin, jotka voidaan lisätä järjestelmään lennosta.

Discovery on pääasiallisesti käyttöliittymä indeksoidun tietokannan (Solr) päälle. Se tarjoaa suuremman rajapinnan verkkokäyttöliittymille tuotetietojen hakemiseen.

Discovery toimii myös välimuistina asiakaspuolen datalle, jotta voidaan erottaa loppukäyttäjän toiminnot palvelinpuolesta. Solr tarjoaa siis nopean pääsyn asiakaspuolen toteutuksille listata ja hakea tuotteita.

Kuva 3 havainnollistaa, kuinka loppukäyttäjät ovat suoraan yhteydessä Discovery-komponenttiin ja Publishing-komponentin käytön eli datan lisäyksen ja päivityksen hoitaa palvelun ylläpitäjä. Käytännössä loppukäyttäjä siis suorittaa käyttöliittymässään haun esimerkiksi jollakin hakusanalla, ja siten ottaa yhteyden Discovery-komponenttiin. Ylläpitäjä voi tietysti myös suorittaa hakuja Discoveryn kautta. Kuvassa on ainoastaan tämän työn kannalta tärkeimmät komponentit, mutta todellisuudessa DCS sisältää paljon muitakin komponentteja.



Kuva 3. Publishing- ja Discovery-komponenttien käyttö.

3.2 Scala-ohjelmointikieli ja Scalatra-sovelluskehys

Scala on JVM-kieli, joka sisällyttää sekä olio-ohjelmoinnin että funktionaalisen ohjelmoinnin todella suppeaksi, loogiseksi ja erittäin tehokkaaksi kieleksi [2]. Scalan nimi tulee sanoista scalable (skaalautuva) ja language (kieli). Se integroi sulavasti olio-ohjelmointi- ja funktionaalisten kielien ominaisuuksia [3].

Scala on puhtaasti olio-ohjelmointikieli siinä mielessä, että jokainen Scalan arvo (value) on objekti eli olio. Olioiden tyypit ja käyttäytyminen kuvataan luokkien (class) ja piirteiden (trait) avulla. Luokkia voidaan laajentaa Scalassa aliluokittamalla. [3.]

Scala on funktionaalinen ohjelmointikieli, mutta se ei pakota ohjelmoimaan funktionaalisesti. Scalaa on siis mahdollista kirjoittaa lähes täysin funktionaalisesti ja myös yhdistämään siihen olio-ohjelmoinnille tyypillisiä ohjelmointityylejä. Scala tarjoaa kevyen syntaksin anonyymifunktioiden määrittelemiseen, tukee korkeamman tason funktioita, mahdollistaa sisäkkäiset funktiot, ja tukee currying-menetelmää. Scalan singleton-oliot tarjoavat kätevän tavan ryhmittää toimintoja, jotka eivät ole luokan jäseniä. Scala tukee myös laiskaa evaluointia, joka on myös yksi yleisistä funktionaalisen ohjelmointikielen ominaisuuksista. [3.]

Lisäksi Scala on erityisen hyvä ohjelmointikieli XML-datan prosessointiin, mikä tekee Scalasta ideaalin webpalvelujen kehittämiseen. Tästä syystä DCS-projektissa dataa käsitelläänkin Scalalla XML-muodossa.

Scalatra on yksinkertainen, helppokäyttöinen ja ilmainen asynkroninen verkkosovelluskehys (framework). Se yhdistää JVM:n voiman Scalan tyylikkyyteen ja suppeuteen, mikä auttaa rakentamaan nopeasti tehokkaita verkkosovelluksia ja ohjelmointirajapintoja. Scalatra on helposti ymmärrettävä tapa tehdä verkkojärjestelmiä, jotka omaksuvat HTTP:n tilattoman luonteen. Scalatran avulla voi rakentaa lähes mitä vaan, mutta yksi sen suurimmista vahvuuksista on REST-ohjelmointirajapintojen rakentaminen. [4.]

Esimerkkikoodissa 1 on havainnollistettu minimaalinen kokonainen toimiva Scalatra-sovellus, joka on vain 7 riviä koodia. Scalatrassa on pyritty minimalisoimaan kirjoitettavan koodin määrä ja samalla tekemään koodista helpommin luettavaa.

```
package com.example.app
import org.scalatra._

class MyScalatraServlet extends ScalatraServlet {
      get("/") {
            <html>
            <body>
            <h1>Hello, world!</h1>
            </body>
            </html>
      }
}
```

Esimerkkikoodi 1. Minimaalinen Scalatra-verkkosovellus.

Esimerkkikoodissa on määritelty yksi metodi, joka on HTTP GET -pyyntö polkuun ”/”. Sitä kutsuttaessa se palauttaa vastauksena HTML:ää, jonka otsikossa lukee ”Hello

world!”. Tällaiset GET-pyynnöt voivat palauttaa dataa myös muun muassa XML-formaatissa tai tavallisena tekstinä. Tämä koodi voidaan ajaa Java Servlet Containereissa esimerkiksi Jettyssä (oletusarvoisesti portissa 8080), jolloin sitä voidaan käyttää lähettämällä GET-pyynnön osoitteeseen http://localhost:8080/.

3.3 PostgreSQL-tietokantajärjestelmä

PostgreSQL on tehokas avoimen lähdekoodin olio-relaatiotietokantapalvelin eli tietokannan hallintajärjestelmä. Se julkaistiin jo vuonna 1996, ja se on saavuttanut vahvan maineen luotettavuudesta, tietojen eheydestä ja oikeellisuudesta. Se toimii kaikissa tärkeimmissä käyttöjärjestelmissä, kuten Linuxissa, UNIX:issa (AIX, BSD, HP-UX, macOS, Solaris jne.) ja Windowsissa. [5.]

Olio-relaatiotietokantojen perustavoitteena on luoda yhteys relaatiotietokantojen ja oliopohjaisten (object-oriented) mallintamismenetelmien välille. Olio-pohjaisia mallintamismenetelmiä käytetään ohjelmointikielissä kuten Java, C++, .NET ja C#. PostgreSQL:n yksi parhaista ominaisuuksista on tyyppien määritteleminen. Datan muunto relaatiomallista oliomalliin ja toisin päin on ongelmallista. PostgreSQL pyrkii helpottamaan tätä tarjoamalla mahdollisuuden rakentaa omia kustomoituja tietotyyppejä.

PostgreSQL ja muut ORD-tietokannat (object-relational database) ovat ORM-kehyksille (object-relational mapping) vaihtoehtoinen ratkaisutapa oliopohjaisen datan tallentamiseen. ORM on ohjelmointitekniikka datan muuntamiseksi yhteensopimattomien tyyppijärjestelmien välillä käyttäen oliopohjaisia ohjelmointikieliä. Tämä luo käytännössä virtuaalisen oliotietokannan, jota voidaan käyttää ohjelmointikielellä. ORD on relaatiotietokannan kaltainen tietokannanhallintajärjestelmä, mutta oliopohjaisella tietokantomallilla eli olioita, luokkia ja perimistä tuetaan suoraan tietokannan skeemoissa ja kyselykielessä. ORM ja ORD tarjoavat siis ratkaisun samaan ongelmaan, mutta eri tavoin.

PostgreSQL tukee viiteavaimia (foreign key), liitoksia (joins), herättimiä (triggers), näkymiä (views) ja tallennettuja proseduureja (stored procedures). Se sisältää yleisimmät SQL-tietotyypit, kuten INTEGER (kokonaisluku), NUMERIC (numeerinen), BOOLEAN (totuusarvomuuttuja), CHAR (kiinteäpituinen teksti), VARCHAR (vaihtuvapituinen teksti), DATE (päiväys), INTERVAL (aikaväli) ja TIMESTAMP

(aikaleima). PostgreSQL tukee myös suurien objektien, kuten kuvien, äänien ja videokuvan tallentamista. Se sisältää ohjelmointirajapinnat muun muassa C/C++-, Java-, .Net-, Perl-, Python-, Ruby-, Tcl- ja ODBC-kielille. [5.]

DCS-projektissa tietokannan hallintajärjestelmäksi on valittu PostgreSQL, koska se on täysin avoimen lähdekoodin toteutus, eli sitä ei omista kukaan eikä sen käyttöön tarvita lisenssisopimusta. Esimerkiksi PostgreSQL:n yhden suurimman kilpailijan MySQL:n käyttöön tuotannossa vaaditaan lisenssi, sillä sen omistaa Oracle. PostgreSQL tarjoaa kaikki ominaisuudet, joita DCS-projektissa tarvitaan ja paljon enemmänkin.

PostgreSQL on erittäin suosittu ja yksi käytetyimmistä tietokantaratkaisuista. Sen suosio johtuu ominaisuuksiensa lisäksi myös siitä, että sen käyttöön ei tarvita lisenssisopimuksia. PostgreSQL:ää käyttää muun muassa Cisco, Skype, Apple, Sun Microsystems, Yhdysvaltain ulkoministeriö sekä useat muut yritykset, valtioiden virastot ja yliopistot ympäri maailmaa [5].

3.4 Apache Tomcat -verkkopalvelin

Apache Tomcat on avoimen lähdekoodin toteutus Java Servlet-, JavaServer Pages-, Java Expression Language- ja Java WebSocket -teknologioista. Tomcat on siis verkkopalvelin, jota kehitetään avoimessa ympäristössä ja joka on julkaistu Apache-lisenssiversion 2 alla. [6.]

Tomcatin tehtävä on suorittaa Java Servlettejä ja renderöidä verkkosivuja, jotka sisältävät JavaServer Pages -koodia. Lyhykäisyydessään Tomcat saa Java-verkkosovellukset toimimaan isäntä- (host) ja palvelinpohjaisilla järjestelmillä. Se tarjoaa ”puhtaan” Java HTTP -verkkopalvelimen, jossa ajaa Java-koodia. Tomcatia voidaan käyttää joko itsenäisenä tuotteena, jolla on oma sisäinen verkkopalvelin tai muiden verkkopalvelimien, kuten Apachen, Netscape Enterprice Serverin, Microsoft Internet Information Serverin tai Microsoft personal Web Serverin kanssa. Tomcat vaatii Java Runtime Enterprise Environment -ympäristön toimiakseen. [7.]

Tomcat koostuu kolmesta pääkomponentista:

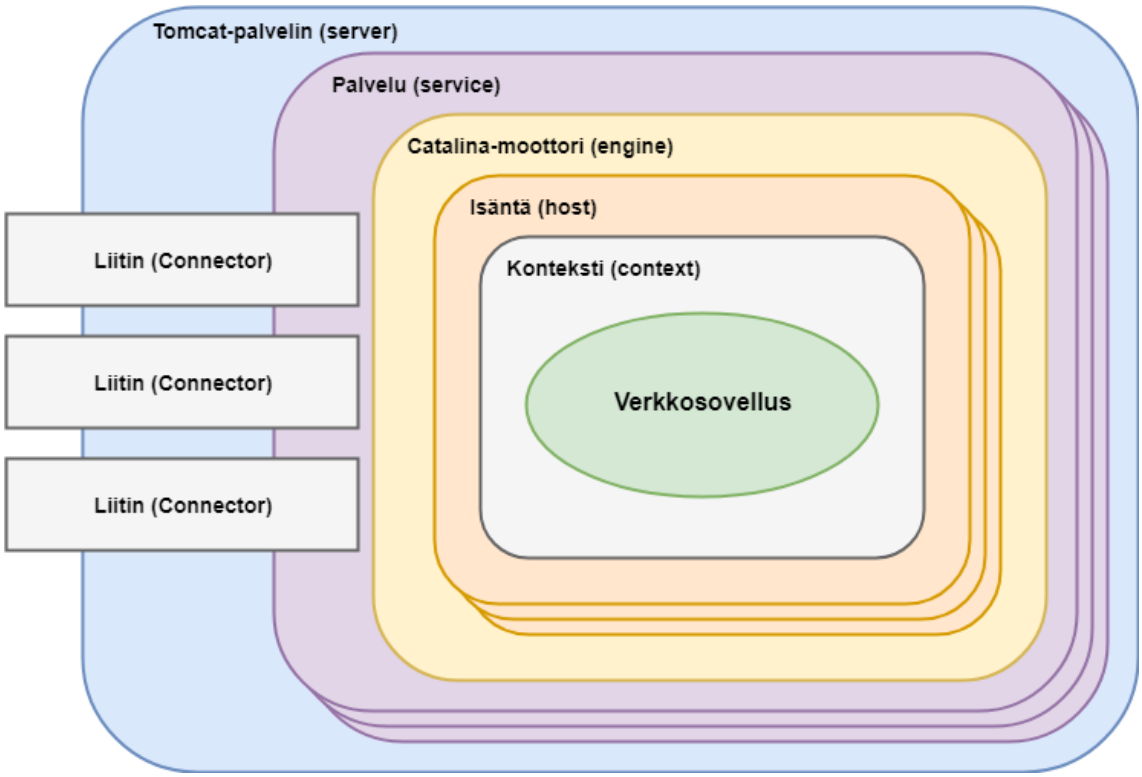
1. Catalina

2. Coyote
3. Jasper.

Catalina on Tomcatin Servlet Container eli servlettien ajoympäristö. Coyote on liitinkomponentti (connector) Tomcatille, ja tukee HTTP-protokollaa. Tämä sallii Catalinan toimia myös tavallisena verkkopalvelimena. Jasper on Tomcatin JSP-moottori, joka mahdollistaa JavaServer Pages -sovelluksien ajamisen Tomcatissä. [8.]

Cluster on klusterikomponentti, jonka tehtävä on hallita isokokoisia sovelluksia. Cluster toimii kuormanjakajana (load balancer), joka jakaa vaadittua työmäärää niin, että palvelimien kuormitus on tasainen. Tätä kutsutaan klusteroinniksi. Klusterointi on tärkeää, jotta palvelimia on aina tarjolla loppukäyttäjille. Esimerkiksi jos palvelimen maksimi kapasiteetti on 5000 käyttäjää, 5001. käyttäjä ja kaikki sen jälkeen saavat virheilmoituksen, eivätkä pysty yhdistämään palvelimeen. Tähän tarvitaan kuormanjakajaa, joka siirtää käyttäjiä palvelimille, joissa on tilaa. Samoin myös, jos yksi palvelin kaatuu, kuormanjakaja voi ohjata liikenteen toiselle palvelimelle. [9.]

Kuvassa 4 on havainnollistettu Tomcat-palvelimen arkkitehtuuria. Tomcat toimii Windows-palveluna (Windows service) tai Linux- tai Unix -daemonina (palveluprosessina), joka odottaa yhteyksiä oletusarvoisesti portista 8080. Yksi Tomcat-instanssi voi tarjota useita palveluita (service), vaikka se onkin epätavallista. Jokaisella Tomcat palvelulla on vähintään yksi (ja mahdollisesti useampi) liitin (connector), ja vähintään yksi (ja mahdollisesti useampi) kehys (container), jossa Catalina-moottori tarjoaa palvelun. [7.]



Kuva 4. Tomcatin arkkitehtuuri.

Esimerkiksi jos yrityksellä on Java-pohjainen sovellus, jonka he haluavat ottaa käyttöön, he voivat ottaa käyttöön Tomcat-palvelimen, joka pystytetään fyysiselle laitteelle. Kun Tomcat on toiminnassa fyysisellä laitteella, voidaan sovellus ottaa käyttöön (deploy) Tomcatissa. Tällöin sovellus toimii verkkosovelluksena ja siihen voidaan ottaa yhteys.

3.5 Apache Solr -hakualusta

Apache Solr on skaalautuva avoimen lähdekoodin tekstikeskeinen hakualusta, joka on rakennettu Apache Lucenen päälle [1]. Olipa kyseessä big datan käsittely, monipuolisen verkkosovelluksen kehittäminen tai pilvipalveluiden rakentaminen, on tärkeää, että sillä on nopea ja luotettava hakuratkaisu. Apache Lucene on ilmainen avoimen lähdekoodin hakukoneohjelmistokirjasto. Ennen versiota 5.0 Solr oli Java-verkkosovellus, joka toimi missä tahansa modernissa Java Servlet -kehyksessä. Näitä ovat esimerkiksi Jetty ja Tomcat, jota käytettiin DCS-projektissa. Versiossa 5.0 tämä kuitenkin muuttui, jonka jälkeen Solr on toiminut omana erillisenä palvelimenaan. Vaikka Solr onkin Java-pohjainen palvelin, se tarjoaa ohjelmointirajapintansa myös ainakin PHP-, Python-, C#-, Ruby- ja .NET-kielillä.

Solr on NoSQL-tekniikka, joka tarjoaa keskeisiä ominaisuuksia, kuten avainsanoilla hakemisen useilla eri kielillä, faceted search -ominaisuuden, sisällön klusteroinnin ja merkityksellisyyden arvioinnin. NoSQL (non SQL tai Not only SQL) on tietokantaratkaisu, jonka toimintaperiaate ei perustu relaatiotietokantamalliin, mutta voi tukea myös joitain relaatiotietokannoille tyypillisiä SQL-tyylisiä hakukieliä. Edellä mainittuja Solrin ominaisuuksia käydään läpi tarkemmin myöhemmin. Solr tarjoaa myös oikeinkirjoituksen tarkistamisen, synonyymikäsittelyn, lausekyselyjä ja tekstianalyysityökaluja kyselyn (query) kielellisten vaihteluiden käsittelyyn. Solr on myös tehokas ratkaisu paikkatietojen (geospatial) kyselyiden toteuttamiseen. Solrin paikkatietotuen avulla voi lajitella asiakirjoja myös maantieteellisen etäisyyden perusteella. [1.]

Solrissa on myös oma oikolukuominaisuus. Solrin oikolukuohjelma tukee kahta perusmuotoa: Auto-correct ja Did you mean [1]. Auto-correct muotoa voidaan käyttää Solrissa automaattisesti tilanteissa, joissa haettua sanaa ei löydy indeksistä. Tällöin Solr hakee dataa korjaamallaan sanalla, jolla löytyy dataa indeksistä. Did you mean -vaihtoehdossa Solr voi ehdottaa käyttäjälle vaihtoehtoista lähes samalla tavalla kirjoitettavaa sanaa, joka löytyy indeksistä. Solr ja Lucene on rakennettu alusta asti tukemaan useita eri kieliä. Solrilla on sisäänrakennettu kielentunnistus, ja se tarjoaa kielikohtaisia tekstin analysointiratkaisuja monille kielille.

Solr on optimoitu etsimään suuria määriä tekstikeskeistä dataa, mutta sitä voidaan käyttää myös esimerkiksi numeroilla tai päivämäärillä hakemisessa. Se mahdollistaa myös jo olemassa olevien indeksoitujen asiakirjojen muokkaamisen.

Keskeinen ero Solrin ja relaatiotietokannan SQL-kyselyiden välillä on se, että Solr lajittelee tiedostot merkityksellisyyden mukaan, kun taas relaatiotietokannan tulokset voidaan lajitella vain yhden tai useamman taulun sarakkeen perusteella. Toisin sanoen, asiakirjojen luokittelu merkityksellisyyden perusteella on avainominaisuus tiedonhaussa ja se auttaa erottamaan Solrin muista kyselytyyleistä. [1.]

Vaikka Apache Solr onkin erittäin tehokas hakualusta, on sillä kuitenkin rajoitteensa. Yksi näistä on se, ettei Solr ole millään tavoin suhteessa asiakirjoihin. Se ei sovellu merkittävien tietomäärien liittämiseen (joining) eri asiakirjojen eri kenttien välillä, eikä se voi suorittaa liittämistoimintoja (join-operations) lainkaan useiden palvelimien välillä. Tällainen olettamus asiakirjojen riippumattomuudesta on monien NoSQL-teknologioiden yleinen kompromissi, koska se mahdollistaa niiden skaalautuvuuden yli

relaatiotietokantojen rajojen. Samasta syystä aiheutuu toinenkin ongelma: tarpeettomat tiedot on toistettava jokaisen asiakirjan osalta, joita kyseiset tiedot koskevat. Tämä voi olla erityisen ongelmallista, kun yhden kentän tiedot, jotka jaetaan useissa asiakirjoissa, muuttuvat. [1.]

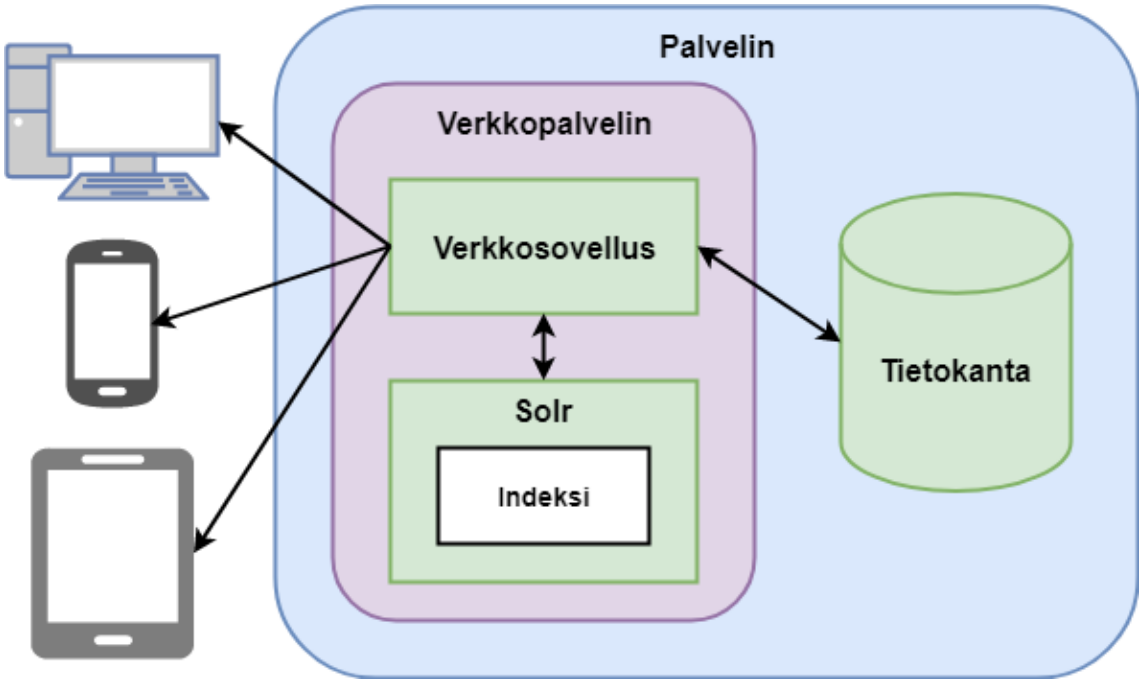
Solr on optimoitu tietylle käyttötapaukselle, joka on hakukyselyiden vastaanottaminen pienillä määrillä hakutermejä, etsiä nopeasti jokaista termiä vastaavat asiakirjat, laskea merkityksellisyys ja järjestää ne sen mukaan ja palauttaa sitten vain muutama tulos näytille. Solria ei ole optimoitu pitkien kyselyiden (tuhansia termejä) käsittelemiseksi tai palauttamaan suuria hakutulosjoukkoja käyttäjille.

Apache Solr on todella suosittu hakualusta, sillä se on nopea, se tarjoaa paljon ominaisuuksia useisiin eri tarpeisiin ja se on avoimen lähdekoodin toteutus, jota voi kuka vain käyttää ilmaiseksi. Vuonna 2012 sitä käyttivät ohjelmistoissaan muun muassa Netflix, Nasa, Instagram, Ebay, Apple ja Sears. Esimerkiksi Netflix käytti Solria sivuston hakutoimintoon ja Sears hakutoimintoon, ennakoivaan hakuun ja faceted navigointiin. [10.]

4 Apache Solr -hakualustan toiminta

Solr tarjoaa REST-tyyppisen API:n, jonka kautta saadaan yhteys Solr-palvelimeen. API (Application programming interface) eli ohjelmointirajapinta on kokoelma valmiita toimintoja, joiden avulla voidaan luoda sovelluksia, jotka käyttävät toisen palvelun ominaisuuksia tai dataa. Solr on verkkosovellus, joten keskusteleminen API:n kanssa tapahtuu HTTP-protokollan avulla. Solrin HTTP API ei kuitenkaan noudata täysin kaikkia REST:in periaatteita. Esimerkiksi Solr käyttää HTTP DELETE -metodin sijaan POST-metodia tiedostojen poistamisessa.

Kuva 5 kuvastaa tyypillistä Apache Solrin toimintaympäristöä (ennen versiota 5.0), jossa verkkosovellus pyörii verkkopalvelimella (esim. Tomcat) ja verkkosovellukseen saa yhteyden tietokoneilla, älypuhelimilla sekä tableteilla. Myös Solr pyörii samalla verkkopalvelimella, sillä sekin on verkkosovellus.



Kuva 5. Esimerkki Solrin toimintaympäristöstä ennen versiota 5.0, jolloin Solr toimi erillisenä verkkosovelluksena samalla verkkopalvelimella.

4.1 Indeksointi ja datan tallentaminen

Indeksointi eli asiasanoitus tarkoittaa tiedon järjestämistä tietyn skeeman tai suunnitelman mukaisesti. Solr pystyy saavuttamaan nopean tiedonhakemisen, sillä sen sijaan, että se etsisi haluttua dataa tekstistä suoraan, se luo indeksitietorakenteita ja etsii niistä. Lisäämällä sisältöä indeksiin siitä tulee haettavaa dataa Solrille. Solr-indeksi voi vastaanottaa tietoja useista eri lähteistä, kuten XML-, JSON- ja CSV-tiedostoista, tietokannan taulukoista poimituista tiedoista sekä tavallisista tiedostomuodoista, kuten Microsoft Wordista tai PDF-tiedostoista. Solr käyttää Lucenen käänteistä indeksointia (inverted index) avukseen tietojen tallentamisessa, mikä mahdollistaa datan hakemisen mahdollisimman nopeasti pelkällä hakusanalla. [1.]

Itse indeksointiprosessi on yksinkertainen, ja se koostuu kolmesta vaiheesta. Ensimmäiseksi data tulee muuntaa Solrin tukemaan formaattiin, joista yleisimmät ovat XML ja JSON. Toisessa vaiheessa data lähetetään Solrille tyypillisesti HTTP POST -metodia käyttäen. Kolmanneksi Solr konfiguroidaan tekemään muutoksia datan tekstiin indeksoinnin aikana. Nämä muutokset ovat siis esimerkiksi isojen kirjainten muuttaminen pieniksi ja sanojen erottaminen toisistaan.

Solr lisää dataa indeksiin pilkkomalla tekstit ensin osiin (facets). Kaikki tekstin sanat erotellaan toisistaan välimerkkien kohdilta, jonka jälkeen tekstit muutetaan sisältämään pelkkiä pieniä kirjaimia. Sanat tallennetaan aakkosjärjestyksessä tauluun, joka kertoo, mihin asiakirjaan ne kuuluvat. Sanat voivat kuulua useampaan tiedostoon samaan aikaan. Jos siis loppukäyttäjä hakee hakusanalla, joka on useammassa eri tiedostossa, voidaan hänelle näyttää kaikki vaihtoehdot, joissa kyseinen sana esiintyy. Taulukossa 1 on havainnollistettu yksinkertainen esimerkki tallennettavasta datasta.

Taulukko 1. Esimerkki indeksiin tallennettavasta datasta.

Tiedosto #	Sisältö
1	Radio App
2	Car App
3	Navigation Helper App
4	Parking Helper
5	Easy Parking

Taulukossa 1 kuvattu data tallennetaan Sorin avulla käänteiseen indeksiin. Taulukko 2 esittää käänteiseen indeksiin tallennettua taulukon 1 dataa.

Taulukko 2. Esimerkki käänteiseen indeksiin tallennetusta datasta.

Termi	Tiedosto #
app	1,2,3
car	2
easy	5
helper	3,4
navigation	3
parking	4,5
radio	1

Kuten taulukosta 2 nähdään, jokainen sana on tallennettu erikseen ilman isoja kirjaimia ja jokainen kuuluu johonkin tiedostoon. Taulukosta huomataan myös, että sanat on tallennettu aakkosjärjestyksessä. Jos loppukäyttäjä hakisi tästä esimerkistä dataa app-hakusanalla, Solr palauttaisi tiedostot 1,2 ja 3.

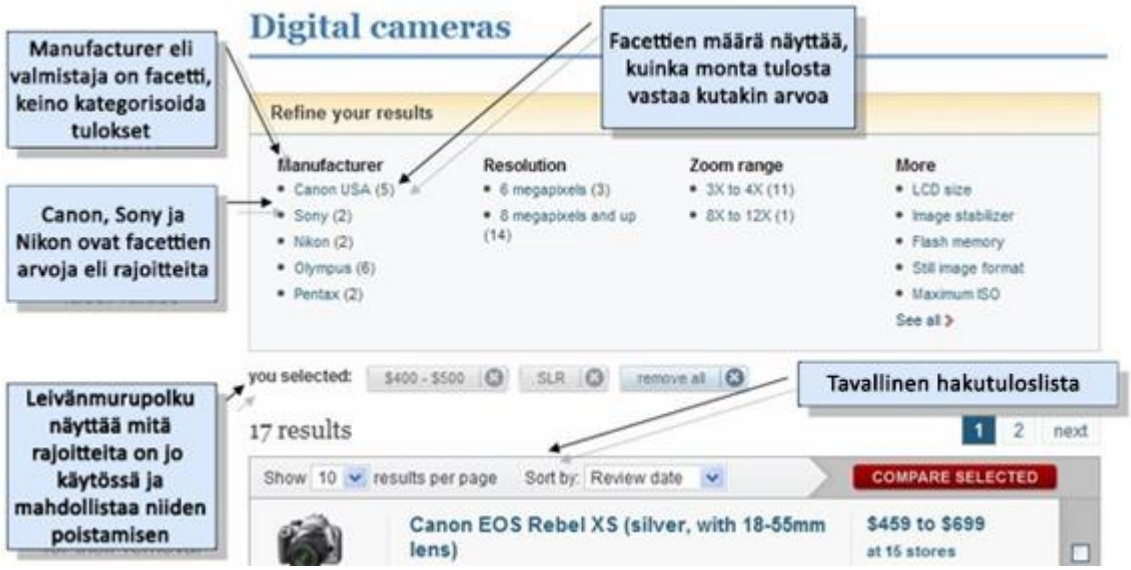
4.2 Datan hakeminen

Solr mahdollistaa datan hakemisen yhdellä tai useammalla hakusanalla. Kuten luvussa 4.0 jo mainittiin, keskusteleminen Solrin kanssa tapahtuu HTTP-protokollan avulla. Hakujen tuottaminen tapahtuu siis yksinkertaisia HTTP GET -pyyntöjä lähettämällä Solr-

palvelimelle. Solr tarjoaa hyviä käyttäjäkokemusta parantavia ominaisuuksia datan haussa. Näistä parhaimpia ovat faceting (luokittelu), ennakoiva haku ja sivutus (pagination).

Faceting eli luokittelu ominaisuuksien mukaan on yksi Solrin tehokkaimmista ominaisuuksista. Faceted search (luokiteltu haku), jota kutsutaan myös nimillä faceted navigation ja faceted browsing, on hakutulosten dynaaminen klusterointi luokkiin, joiden avulla käyttäjät voivat syventyä hakutuloksiin minkä tahansa arvon mukaan missä tahansa kentässä. Faceting tarjoaa käyttäjille työkaluja tarkentamaan hakukriteereitään ja löytämään lisätietoja luokittelemalla hakutulokset alaryhmiin facettien avulla. Jokaisessa näytettävässä facetissa näkyy myös hakutulosten määrä, jotka kuuluvat kyseiseen ryhmään. Käyttäjät voivat sitten "porata syvemmälle" soveltamalla tiettyjä rajoituksia hakutuloksiin. [11.]

Kuvassa 6 on havainnollistettu erästä faceted searchin toteutusta, jossa Solrin hakutulokset on jäsennelty ryhmittäin facetteihin ja facettien arvot sijoitettu allekkain. Tässä esimerkissä verkkosovellus on verkkokauppa, joka myy kameroita. Kameran valmistaja on yksi facetti, jonka mukaan hakutuloksia voidaan kategorisoida. Muita facetteja on esimerkiksi resoluutio ja zoomausetäisyys. Canon, Sony ja muut valmistajat ovat facettien arvoja. Sulkuihin merkitty numero kertoo, kuinka monta tulosta vastaa tietyn facetin arvoa. Listan alapuolella näkyy myös leivänmurupolku, joka näyttää, mitä rajoitteita eli facettien arvoja on jo käytössä.



Kuva 6. Esimerkki faceted search (luokiteltu haku) toteutuksesta [12].

Solr mahdollistaa myös ennakoivan haun. Ennakoivassa haussa loppukäyttäjälle ehdotetaan mahdollisia vaihtoehtoja samaan aikaan kuin hän kirjoittaa hakuaan. Ennakoivan haun tärkeimmät ominaisuudet ovat nopeus ja merkityksellisyys. Solr kykenee ehdottamaan loppukäyttäjälle hakuvaihtoehtoja erittäin nopeasti, ja samalla se pystyy lajittelemaan vaihtoehtoja niiden merkityksellisyyden mukaan. Merkityksellisyys voidaan määritellä esimerkiksi aikaisempien hakujen perusteella niin, että ne hakusanat, joita on eniten haettu, esitetään loppukäyttäjälle ensimmäisenä.

Kun dataa on paljon ja hakutulokset ovat suuria, on myös mahdollista käyttää Solrin sivutus- (pagination) ominaisuuden käyttöönottoa. Sen sijaan, että hakutuloksena palautettaisiin kaikki mahdolliset hakutulokset samalle sivulle, voidaan Solr optimoida palauttamaan tietty määrä hakutuloksia yhdelle sivulle. Sivutus auttaa nopeuttamaan datan hakemista, sillä jokainen kysely palauttaa vain pienen osajoukon hakutuloksista.

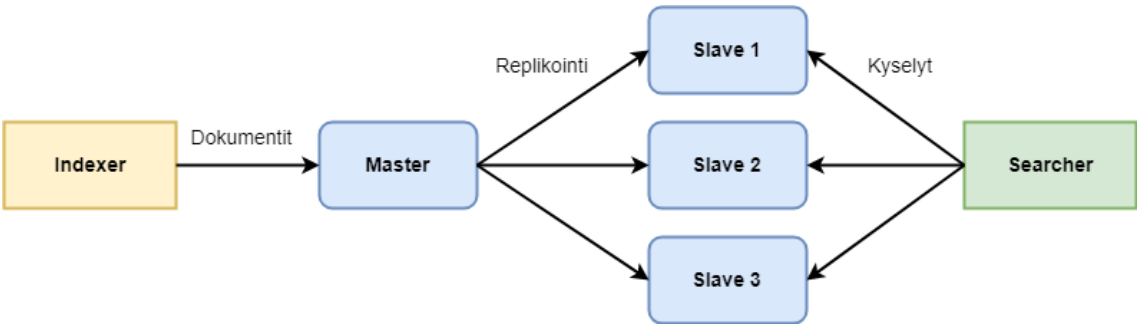
4.3 Solr-indeksin replikointi

Kun hakupyyntöjä tulee paljon, Solr-indeksi kannattaa replikoida (replicate) eli monistaa mahdollistaakseen sujuvan ja nopean hakemisen ja estääkseen palvelimen ylikuormittumisen. Solr-indeksin replikointiin on olemassa kaksi ratkaisumallia: Master-slave (isäntä-orja) ja SolrCloud. Master-slave-mallissa yksi tai useampi Solr-palvelin toimii itsenäisessä tilassa. SolrCloudissa yksi tai useampi Solr-palvelin toimii yhdessä Apache ZooKeeperin kanssa, joka koordinoi niiden toimintaa.

Master-slave-mallia käytettäessä jaetaan master-indeksin täydelliset kopiot yhdelle tai useammalle slave-palvelimelle. Master-palvelin vastaanottaa kaikki päivitykset ja kaikki muutokset (kuten lisäykset, päivitykset tai poistot) tehdään yhdelle master-palvelimelle. Masteriin tehdyt muutokset jaetaan kaikille slave-palvelimille, jotka käsittelevät kaikki kyselypyynnöt. Tämä työnjako mahdollistaa sen, että Solr skaalautuu riittävästi reagoidakseen suureen määrään hakukyselyitä. Master-palvelin seuraa malleja, metatietoja, käyttöoikeuksia ja sisältöä, kun taas slave-palvelin seuraa vain malleja. [13.]

Kuvassa 7 on havainnollistettu esimerkki master-slave-mallista, jossa Indexer eli DCS:n tapauksessa Publishing-komponentti lähettää dataa master-palvelimen indeksiin ja master replikoi indeksinsä kolmelle slave-palvelimelle, jotka vastaanottavat kaikki

kyselyt. Kyselyt lähetetään DCS-projektissa Discovery-komponentin kautta. Tämä on yksinkertainen ja suoraviivainen replikointimalli.



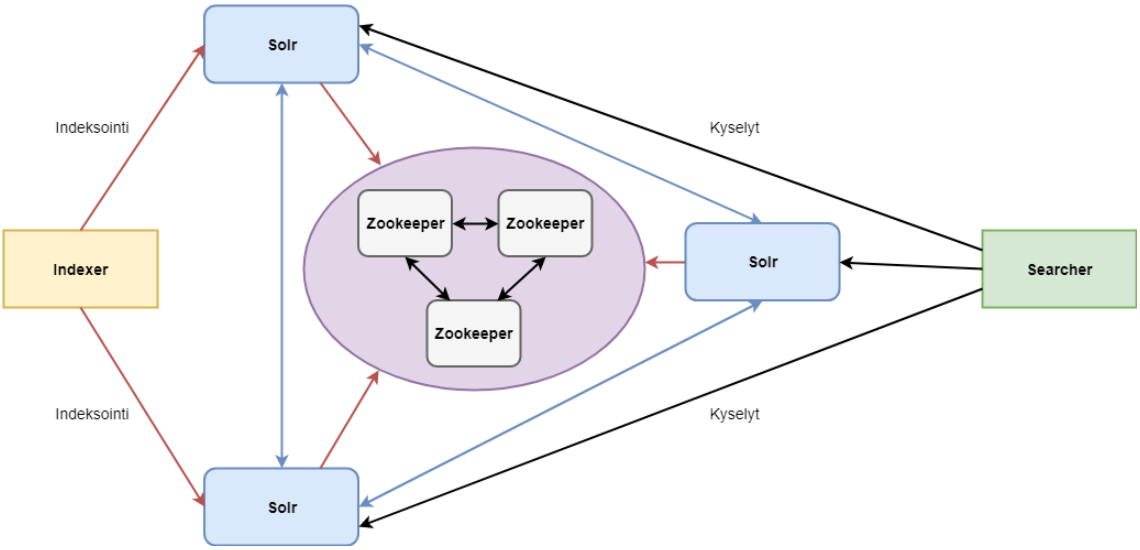
Kuva 7. Esimerkki master-slave-mallin toimintaympäristöstä.

SolrCloud ajaa saman asian, mutta tarjoaa enemmän ominaisuuksia erilaisella rakenteella. SolrCloud on luotu korvaamaan aikaisempi master-slave-malli, mutta jos sen tarjoamia ominaisuuksia ei tarvita, voidaan hyvin käyttää master-slave-mallia. SolrCloud toimii ilman isäntäsolmua (master node) jakamassa solmuja (node), sirpaleita (shards) ja replikaatioita. Sen sijaan Solr käyttää Apache ZooKeeperia hallitsemaan näitä sijainteja konfigurointitiedostoista ja kaavioista riippuen. Kyselyt ja päivitykset voidaan lähettää mille tahansa palvelimelle. Solr käyttää ZooKeeperin tietokannan tietoja selvittääkseen, minkä palvelimien täytyy käsitellä pyyntö. [14.]

Ennen SolrCloudin toiminnallisuuksien opettelua täytyy ymmärtää muutama sen käsite. Solrissa termiä core (ydin) käytetään viitatessa yhteen indeksiin. Solr-palvelimella voi olla yksi tai useampi core. Node eli solmu on JVM-instanssi eli ilmentymä, joka ajaa Solria. Toisin sanoen node on palvelin. Myös Solr corea voidaan kutsua nodeksi. Jokainen node voi sisältää sekä Solr-instanssin että monenlaista dataa. SolrCloudissa indeksit voidaan halutessa jakaa paloiksi, joita kutsutaan sirpaleiksi eli shardeiksi. Jokainen shard on itsessään täysin toimiva ja itsenäinen Solr-indeksi. Jokainen node voi siis sisältää useita shardeja. SolrCloudissa ei ole master- tai slave-palvelimia. Sen sijaan jokainen shard koostuu ainakin yhdestä fyysisestä kopiosta, joista yksi on johtaja (shard leader). Shard leaderin päätehtävä on vastaanottaa muutokset ja lähettää ne myös replikaatioille. Shard leaderin valitsee automaattisesti Zookeeper, eikä sillä juurikaan ole merkitystä, mikä shardeista on johtaja.

Kuva 8 havainnollistaa esimerkkiä SolrCloud-mallin mukaan rakennetusta toimintaympäristöstä. Kuvassa esitetyssä ratkaisussa on käytetty kolmea Solr-solmua.

SolrCloudissa päivityspyynnöt voidaan lähettää klusterin mille tahansa solmulle, ja pyyntö toimitetaan solmun shard leaderille. Shard leader indeksoi datan paikallisesti ennen sen lähettämistä replikaatioille. Myös kyselyt voidaan lähettää mille tahansa solmulle. Kun Solr-solmu vastaanottaa hakupyynnön, pyyntö reititetään automaattisesti replikaatiolle, joka on osa etsittävää kokoelmaa.



Kuva 8. Esimerkki SolrCloud-mallin toimintaympäristöstä.

Kuten siis huomataan, SolrCloud on monimutkaisempi kuin master-slave-malli, mutta myös vikasietoisempi. SolrCloud on erityisesti silloin parempi vaihtoehto, kun dataa on erittäin paljon tai kun dataa halutaan järjestää useisiin eri indekseihin niiden eriävyyksien vuoksi.

5 Apache Solrin päivittäminen

Apache Solrin päivittäminen oli päätehtäväni insinöörityössäni. Tarkoituksena oli päivittää DCS-projektissa käytössä oleva Solr versiosta 3.4 uusimpaan versioon, joka sillä hetkellä oli 7.3. Päivitys haluttiin tehdä, sillä käytössä oleva versio oli paljon jäljessä uusinta versiota.

Ennen kuin varsinaisia muutoksia pääsi tekemään, täytyi ensin käydä läpi, kuinka Solr oli toteutettu DCS-projektissa. Solr oli toteutettu master-slave-mallin mukaisesti, sillä se oli ainut replikoinnin ratkaisumalli ennen 4.0-versiota. Se oli myös täysin riittävä ratkaisu DCS-projektiin, koska indeksoitavaa dataa ei ollut erityisen paljon. Solria ajettiin Tomcat-

verkkopalvelimella. Projektissa dataa käsiteltiin pääosin aina XML-muodossa, ja Solr palauttikin kyselyiden vastaukset oletusarvoisesti XML:nä.

Tietenkin piti myös selvittää, mitä oli muuttunut Solrin päivityksissä. Yksi suurimmista muutoksista tapahtui versiossa 4.0, jolloin SolrCloud julkaistiin. SolrCloud on kuitenkin vain vaihtoehto aikaisemmalle master-slave-mallille ja päätimme olla vaihtamatta SolrCloudiin päivityksen yhteydessä, koska sille ei ollut tarvetta. Versiossa 5.0 tapahtui muutos, jonka jälkeen Solrin ajamista Java Servlet -palvelimella, kuten Tomcatissa, ei enää tueta. Tämän seurauksena Solr tuli muuttaa niin, että sitä ajetaan omana palvelimenaan. Version 6.0 muutokset eivät vaikuta suuremmin DCS-projektiin, mutta yksi näistä suurimmista muutoksista oli, ettei Java 8:aa aikaisempia versioita enää tueta. Versiossa 7.0 JSON vaihtui oletusarvoiseksi datan formaatiksi, joka aikaisemmin oli XML. DCS-projektissa datan formaattina käytettiin XML:ää, jonka vaihtaminen JSON:ksi oli tarkoitus tehdä tulevaisuudessa, mutta toistaiseksi sitä ei tarvinnut muuttaa. Tämä tuli kuitenkin huomioida kyselyiden lähettämisessä, sillä tämän muutoksen jälkeen kyselyihin täytyy erikseen asettaa palautusformaatiksi XML. Näiden muutosten lisäksi Solrin Java API:n useita luokkia oli vanhentunut ja ne tuli korvata uusilla luokilla tai muilla menetelmillä.

5.1 Koodin kääntäminen

Aloitin päivittämisen vaihtamalla DCS:ssä käytettävän Solrin uusimpaan versioon, jonka seurauksena kaikki käytössä olevat Solrin luokat, joita ei enää tuettu muuttuivat ohjelmointiympäristössä punaisiksi, joka indikoi virheestä ja siten oli helppo löytää muutoksia vaativat koodit. Etsin näille luokille uudet korvaavat toteutukset, joita löytyi lähes kaikille. Joillekin luokille ei ollut korvaavia luokkia Solrin uusimmassa versiossa, joten täytyi hieman soveltaa, mutta suurempia ongelmia tästä ei syntynyt.

Solr-komponenttiin oli yritetty toteuttaa Embedded Solr -toteutusta, mikä oli jäänyt vaiheeseen. Tätä ei haluttu enää toteuttaa, joten poistin kaikki siihen liittyvät koodit. Tämän lisäksi poistin myös turhaa koodia ja siistin samalla hieman jäljelle jäävää koodia.

Tähän asti kaikki oli helppoa, sillä koodia vasta yritettiin saada sellaiseen muotoon, että se menee kääntäjästä (compiler) läpi. Seuraavaksi tuli luoda testiympäristö, jotta voitiin

suorittaa virheenkorjausta (debugging) ja saada koodi toimimaan Solrin kanssa virheettömästi.

5.2 Testiympäristön luominen

Päivitystä varten minun tuli pystyttää paikallinen testiympäristö omalle tietokoneelleni. Aloitin lataamalla Solrin ja Tomcatin uusimmat versiot tietokoneelleni ja kokeilin, että ne toimivat oikein yrittämällä käynnistää ne.

Tomcat vaati pientä konfiguraatiota ennen käynnistämistä. Loin käyttäjän, jolla oli sama nimi ja salasana kuin DCS-projektissa, jotta sitä ei tarvinnut vaihtaa koodista testausta varten. Kun Tomcat oli konfiguroitu, pystyin käynnistämään sen. Tämän jälkeen Publishing-komponentti tuli ottaa käyttöön (deploy) Tomcatissa, minkä vuoksi jouduin myös luomaan DCS-projektin Publishing-komponenttiin kansion paikallisen testiympäristön properties (ominaisuudet) -tiedostoja varten. Näissä tiedostoissa määrittelin testiympäristön localhostien (paikallisten isäntien) portit, joita tuli käyttää. Tein uudet tiedostot, jotta varsinaisiin tuotannossa käytettäviin properties-tiedostoihin ei tarvinnut koskea. Tämän jälkeen pystyin ottamaan Publishing-komponentin käyttöön Tomcatissa. Sama täytyi tehdä myös Discovery-komponentille. Loin sille kansion paikallisen testiympäristön properties-tiedostoja varten ja vaihdoin url-osotteiden portit testiympäristössä käytössä oleviksi porteiksi. Tämän jälkeen Discovery-komponentti oli valmis käyttöönotettavaksi Tomcatissa.

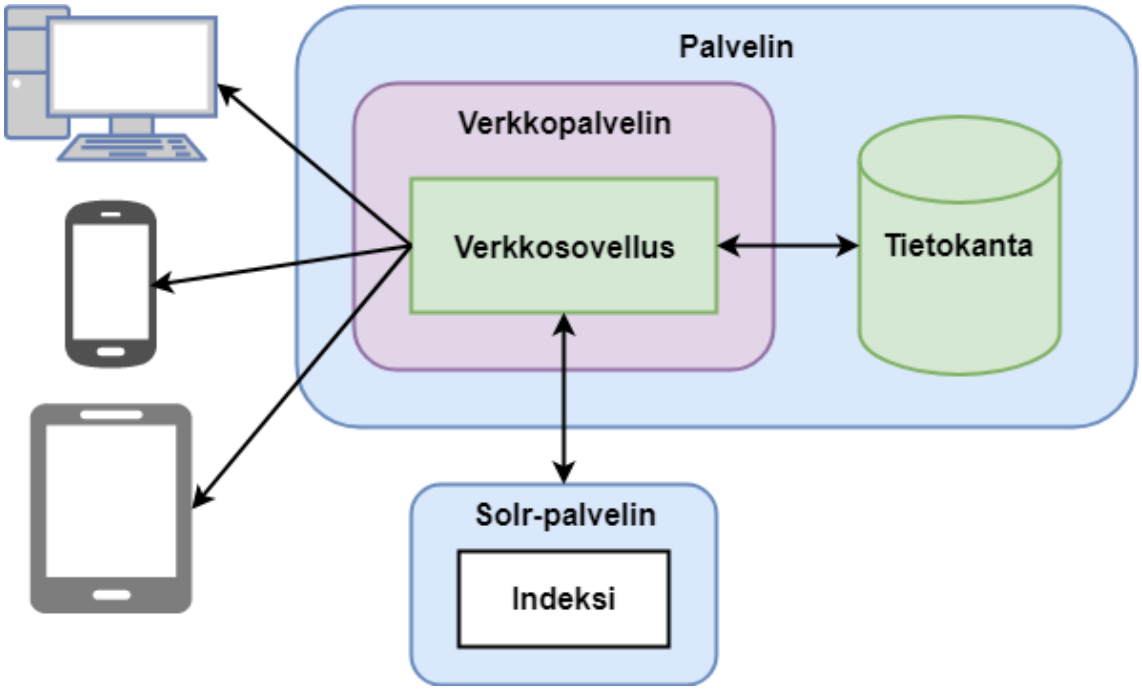
Tietokantana DCS-projektissa toimii PostgreSQL, joten tässä vaiheessa minun tuli myös ladata se tietokoneelleni ja konfiguroida se. Loin PostgreSQL:ään tietokannan (database) ja roolin (role) ja lisäsin roolin omistajaksi kannalle. Publishing-komponentti tallentaa tiedot kyseiseen tietokantaan.

Solr ei vaatinut käynnistyäkseen mitään konfigurointia, mutta toimiakseen DCS-projektin kanssa muutoksia tuli tietenkin tehdä. Toimiakseen Solr tarvitsee vähintään yhden coren eli ytimen. Loin Solrille ytimen ja kopioin ytimelle skeeman, jota DCS käyttää. Skeema on tiedosto, joka esittelee, millaisia kenttiä on, mitkä niistä ovat pakollisia, mitkä niistä ovat uniikkeja ja mitkä pääavaimia ja myös, kuinka indeksoida ja etsiä kutakin kenttää. Tiesin kuitenkin, että skeema ei todennäköisesti tule toimimaan sellaisenaan Solrin uusimman version kanssa, mutta tästä oli hyvä aloittaa, jotta sai virheilmoitukset

näkyviin. Kopioin myös vanhan käytössä olleen Solrin konfiguraatiotiedoston uudelle ytimelle.

Apache Solr 5.0 -version myötä Solrin ajamista Servlet Container -kehyksissä ei enää tueta, joten ympäristöä tuli muuttaa niin, että Solria ajettaisiin omana palvelimenaan. Aikaisemmin Solria ajettiin DCS-projektissa Tomcatissa muiden komponenttien kanssa, mutta päivityksen myötä se tuli siis eristää niistä. Tämä oli hyvin yksinkertaista, sillä muutoksen vuoksi vain Solrin osoite tuli muuttaa, ja koska pystyttämäni testiympäristön tuli toimia paikallisesti omalla tietokoneellani, vain portti 8080 (Tomcatin portti) tuli muuttaa portiksi 8983, joka on Solrin oletusarvoinen portti.

Kuvassa 9 on havainnollistettu esimerkki Apache Solrin toimintaympäristöstä version 5.0 jälkeen Solrin toimiessa omana erillisenä palvelimenaan (vrt. kuva 5 s. 15).



Kuva 9. Esimerkki Apache Solrin toimintaympäristö versiossa 5.0 ja sen jälkeen.

5.3 Virheenkorjaus ja ohjelmointi

Kun testiympäristö oli valmis ja koodit saatu kääntyvään muotoon, Solrin toimintaa varsinaisen koodin kanssa pystyttiin testaamaan ja voitiin siirtyä virheenkorjaukseen (debugging) ja ohjelmointiin.

Ongelmien löytäminen oli helppoa. Heti Solrin käynnistäessä se ilmoitti virheistä skeemassa, jonka olin sille aikaisemmin kopioinut. Useat luokat olivat vanhentuneet ja ne tuli korvata uusilla. Muutoksia tuli tehdä skeemaan niin paljon, että olisi saattanut olla helpompaa ottaa uusin skeemapohja ja rakentaa uusi versio sen päälle. En kuitenkaan etukäteen tiennyt, paljonko muutettavaa skeemassa on, joten tein päivityksen yksi muutos kerrallaan. Käynnistäessäni Solrin se kertoi aina virheen, johon se kaatui ja etsin internetin avulla ratkaisun sen korjaamiseen. Myös skeeman rakenne oli muuttunut hieman, eikä esimerkiksi types (tyypit) ja fields (kentät) -elementtejä enää tarvittu erittelemään niiden sisältöä. Lopulta Solr käynnistyi uuden skeeman kanssa ja pystyi luomaan ytimen.

Sama ongelma oli myös Solrin konfiguraatiotiedoston kanssa. Iso osa luokista oli vanhentunut ja joidenkin tuki oli poistettu kokonaan ilman korvaavaa toteutusta. Toistin saman prosessin kuin skeeman kanssa eli käynnistin Solria aina uudestaan ja joka kerta sain yhden virheilmoituksen. Etsin ratkaisut virheisiin ja korjasin ne, kunnes Solrin käynnistys onnistui uuden konfiguraatiotiedoston kanssa virheettömästi. Myös konfiguraatiotiedoston rakenne oli muuttunut hieman indeksoinnin konfiguroinnin osalta. Tämä muutos vähensi hieman koodin määrää, sillä se auttoi poistamaan koodia, jota jouduttiin ennen käyttämään konfiguraatiotiedostossa useammin kuin kerran.

Konfiguraatiotiedoston muokkaamisen jälkeen voitiin siirtyä varsinaiseen ohjelmointiin eli Scala-ohjelmointikielellä ohjelmoitujen toiminnallisuuksien muokkaamiseen yhteensopivaksi uuden version kanssa. Muutoksia ei tarvinnut tehdä paljon, mutta eniten muutoksia aiheutti Solrin versiossa 5.0 tullut muutos, jonka jälkeen Solria ei enää kuulunut ajaa Servlet Containerissa. Tarvittavat muutokset olivat kuitenkin yksinkertaisia, sillä riitti, kun poisti CoreContainer-luokan ja siihen liittyvät koodit sekä päivitti käytettävän Solrin URL-osoitteen perään käytössä olevan coren eli ytimen nimen. Näiden muutosten jälkeen kaikki virheilmoitukset Solrin käynnistykseen liittyen katosivat, mutta automaattinen datan lisääminen Solrin indeksiin ei vielä toiminut. Datan lisäämisen DCS-projektissa hoitaa Publishing-komponentti. Ainut muutos, jonka se vaati lisätäkseen dataa Solriin, oli url-osoitteiden porttien muuttaminen. Tämä tuotti hieman ongelmia toisesta komponentista johtuen, mutta port forwardingilla eli portin edelleenohjauksella ongelma ratkesi nopeasti. Näiden muutoksien jälkeen datan lisäys Solriin onnistui, mutta vielä oli jäljellä Discovery-komponentin muokkaaminen, jotta haku Solrista toimisi. Testiympäristön vuoksi myös Discovery-komponentista piti muuttaa muutama url-osoitteen portti. Tämän lisäksi ongelmia aiheuttivat Solrin versiossa 5.0

tulleet muutokset, joiden jälkeen SolrServer-luokka oli vanhentunut ja sen tilalla tuli käyttää SolrClient-luokkaa, mutta pelkkä tämän luokan vaihtaminen ei riittänyt. Jouduin poistamaan yhden metodin, joka aiheutti ongelmia ja toteuttamaan sen ominaisuudet toisella tavalla. Tässä vaiheessa Discovery-komponentti alkoi toimimaan ja sai yhteyden Solriin, mutta haut eivät vielä toimineet täysin oikein. Solrin versiossa 7.0 hakutulosten oletusformaatti vaihtui JSON-muotoon, joten hakukyselyiden luomisen yhteydessä tuli määrittää erikseen, että tulokset halutaan XML-formaatissa. Hakukyselyitä hieman muokkaamalla nekin alkoivat toimimaan ja näin ollen Solrin päivittäminen oli valmis.

6 Hakualustojen vertailu

Työhöni kuului myös hakualustojen vertailu ja mahdollisen korvaajan etsiminen Apache Solrille. Tehtäväni oli siis tutkia, mitä muita hakualustoja on olemassa ja valita niistä muutamia vertailua varten. Valitsin hakualustat vertailuun sen perusteella, mitkä olivat suosituimpia ja mitkä niistä vaikuttivat nopealla perehtymisellä sopivimmilta DCS-projektiin Apache Solrin korvaajaksi.

Hakualustojen vertailua ei tehty siksi, että Apache Solr ei olisi riittävä tai se tuottaisi ongelmia. Solr on osoittautunut hyväksi ja riittäväksi hakualustaksi ainakin DCS-projektissa. Vertailu suoritettiin, jotta Apache Solrille saataisiin vertailukohteita ja mahdollisesti löydettäisiin vielä parempi vaihtoehto. Vertailusta on paljon hyötyä yritykselle myös tulevaisuuden projekteissa, joissa sovellukselle tulee valita hakualusta.

Vertailuissa käydään läpi,

1. mikä kyseessä oleva hakualusta on
2. hakualustan käyttötarkoitus, ominaisuudet ja vahvuudet
3. hakualustan heikkoudet ja puutteet
4. hakualustan suosio ja yritykset, jotka käyttävät hakualustaa
5. vertailu Solriin ja mahdollisesti muihin hakualustoihin.

6.1 Elasticsearch

Mikä on Elasticsearch?

Elasticsearch on suosittu hakualusta, ja se on myös hyvä ensimmäinen vertailukohde, sillä se on erittäin samankaltainen Apache Solrin kanssa ja niitä usein verrataankin toisiinsa. Elasticsearchistä käytetään myös lyhennettä ELK eli Elasticsearch, Logstash ja Kibana, sillä nämä kolme komponenttia toimivat yhdessä muodostaen ohjelmistokokonaisuuden. Elasticsearch on siis todellisuudessa vain yksi komponentti ELK:sta. Elasticsearch on avoimen lähdekoodin hakualusta, joka on rakennettu Apache Lucene -hakukoneohjelmistokirjaston päälle käyttäen Java-ohjelmointikieltä. Elasticsearch toimii itsenäisenä palvelimena ja kaikki kommunikaatio sen kanssa tapahtuu HTTP:n kautta. Elasticsearch tarjoaa ohjelmointirajapintansa muun muassa Java-, JavaScript-, .NET-, PHP-, Scala-, JavaScript- ja Ruby -kielillä. Elasticsearch tukee ainoastaan JSON-dataformaattia datan tallentamisessa ja sen hakemisessa. [15.]

Käyttötarkoitus, ominaisuudet ja vahvuudet

Elasticsearch tukee datan indeksointia samalla tavalla kuin Apache Solr, koska ne molemmat käyttävät Apache Lucenea, joka suorittaa indeksoinnin. Myös replikointi eli palvelinten ilmentymien monistaminen on mahdollista ja se on tehty helppokäyttöiseksi. Datan haku toimii suhteellisen yksinkertaisen Query DSL -kielen avulla. Hakuprosessin periaate on kuitenkin samanlainen Apache Solrin kanssa. Myös Elasticsearch on NoSQL-tekniikka. Se tarjoaa keskeiset ominaisuudet, kuten hakemisen useilla eri kielillä, faceted search -ominaisuuden, ennakoivan haun ja merkityksellisyyden arvioinnin, aivan kuten Apache Solrikin. Elasticsearch tukee datan tallentamisessa lähes kaikkia tiedostotyyppejä. ELK:n Logstash-komponentti tarjoaa myös tuen lokitiedostojen analysointiin. Logstash kerää ja analysoi lokit, jonka jälkeen Elasticsearch indeksoi ja tallentaa tiedot. Kibana on työkalu, joka tarjoaa datan visualisoinnin. Nämä kolme komponenttia toimivat hyvin yhdessä ja muodostavat ELK-ohjelmistokokonaisuuden.

Puutteet ja heikkoudet

Negatiivisia puolia Elasticsearchistä ei ole paljon, mutta yksi näistä on XML-tuen puuttuminen kokonaan. XML on edelleen erittäin laajalti käytetty formaatti, ja jos yritys jostain syystä haluaa käsitellä dataa XML-muodossa eikä halua vaihtaa JSONin käyttöön, on sen syytä harkita jonkun toisen hakualustan käyttöä eli esimerkiksi Apache Solria. Tämä siis koskee vain hakupyyntöjen ja hakutulosten datan käsittelyä eikä tallennettavan dokumentin tiedostomuotoja. Toinen negatiivinen puoli on hieman heikko dokumentaatio varsinkin konfiguraation osalta. Konfiguraation voi tehdä joko JSON-, tai YAML-muodossa, mutta dokumentaatiossa ohjeet ovat sekavat, eikä aina välttämättä ymmärrä kummasta tavasta on kyse.

Suosio ja asiakasyritykset

Elasticsearch on ollut pitkään erittäin suosittu ja se oli ainakin vielä maaliskuussa 2018 kaikista yleisistä hakualustoista suosituin ja eniten käytetty [16]. Sitä käyttävät monet suuret yritykset kuten esimerkiksi Ebay, Netflix, Facebook, Adobe Systems, GitHub, Vimeo, SoundCloud ja Blizzard Entertainment [17].

Vertailu muihin hakualustoihin

Solrilla ja Elasticsearchillä on samat ominaispiirteet ja ne molemmat on rakennettu Apache Lucenen päälle. Juurikin se tekee niistä erittäin samanlaisia varsinkin indeksoinnin ja hakujen osalta. Elasticsearch ja Apache Solr ovat myös molemmat avoimen lähdekoodin hakualustoja ja ominaisuuksien puolesta niissä ei ole merkittäviä eroja. Ominaisuuksien perusteella valitsisin näiden kahden väliltä Elasticsearchin, jos sovellus, jossa hakualustaa tarvitaan, toimisi täysin JSON:in avulla, koska datan käsittely on Elasticsearchissä nimenomaan suunniteltu suoritettavan JSON-muodossa tai jos sovellus tarvitsisi ELK:n Logstash-komponentin tarjoamia ominaisuuksia eli lokitiedostojen analysointia. Jos taas sovellus olisi riippuvainen datan käsittelystä XML-muodossa eikä muita rajoitteita valinnalle olisi, valitsisin Apache Solrin, sillä Elasticsearch ei tue XML-formaattia datan käsittelyssä.

6.2 Splunk

Mikä on Splunk?

Myös Splunk on suosittu hakualusta, mutta sen käyttötarkoitus on hieman erilainen kuin Apache Solrin ja Elasticsearchin. Splunk on ratkaisu laitedatan hyödyntämiselle. Se on helppo, nopea ja joustava tapa kerätä, analysoida ja tallettaa tietojärjestelmien ja teknologiainfrastruktuurien tuottama valtava määrä laitedataa. Splunk kerää ja indeksoi reaaliaikaista dataa hakukelpoiseen arkistoon, josta se voi luoda kaavioita, raportteja, hälytyksiä ja visualisointeja. Splunkin ydintehtävä on lokitiedostojen käsittely. Se voi tallentaa kaikki lokitiedostot ja tarjoaa erittäin nopeat hakutoiminnot, jotka mahdollistavat niiden helpon analysoinnin. Splunk toimii lähes missä tahansa palvelinkäyttöjärjestelmässä ja se tarjoaa HTTP REST API:n, jonka avulla käyttäjät keskustelevat Splunkin kanssa. Splunk tukee C#-, Java-, JavaScript-, PHP-, Python- ja Ruby -ohjelmointikieliä. [18.]

Käyttötarkoitus, ominaisuudet ja vahvuudet

Splunk-hakualusta ei perustu Apache Luceneen eikä mihinkään muuhunkaan kolmannen osapuolen tuotteeseen. Vaikka Splunkissa voi käyttää yksinkertaisia hakutermejä, esim. käyttäjätunnusta ja katsoa, kuinka usein se ilmestyy tietyllä ajanjaksolla, Splunkin hakukäsittelytekniikka (SPL) tarjoaa paljon enemmän. SPL on äärimmäisen tehokas työkalu tarkkailemaan suuria määriä tietoja ja suorittamaan tilastollisia toimintoja siitä, mikä on merkityksellistä tietyssä kontekstissa. Esimerkiksi voidaan selvittää, mitkä sovellukset ovat hitaimpia käynnistymään, eli mitkä niistä aiheuttavat loppukäyttäjälle eniten odotusaikaa. Indeksointivaiheen aikana, kun Splunk käsittelee tallennettavaa dataa ja varautuu sen varastointiin, indeksoija tekee merkittävän muutoksen: se pilkkoo merkkivirran yksittäisiin tapahtumiin (events). Tapahtumat vastaavat tavallisesti käsiteltävän lokitiedoston rivejä. Jokainen tapahtuma saa aikaleiman, joka on tyypillisesti jäsennetty suoraan tuloriviltä ja muutamia muita oletusominaisuuksia, kuten alkuperän eli laitteen, josta se on peräisin. Tämän jälkeen tapahtumien avainsanat lisätään indeksitiedostoon nopeuttamaan myöhempiä hakuja ja tapahtuman teksti tallennetaan pakattuun tiedostoon tiedostojärjestelmään. Splunk tallentaa tiedostot tietojärjestelmään, joten se ei vaadi erillistä tietokantaa. Splunk on myös erittäin helposti skaalautuva. Jos yksittäinen Splunk-palvelin ei riitä, voidaan helposti lisätä toinen. Tallennettava data jakautuu automaattisesti tasaisesti ja haut

ohjautuvat kaikkiin Splunk-ilmentymiin niin, että nopeus kasvaa dataa tallentaneiden koneiden lukumäärän mukaan. Jokainen tapahtuma voidaan halutessa myös tallentaa kahdelle tai useammalle Splunk-palvelimelle. Splunk on erittäin helppokäyttöinen ja varsinkin sen käyttöönotto on helppoa. Splunkille voidaan syöttää raakaa dataa, ja se hoitaa loput. Splunk hyväksyy lähes kaiken muotoisen datan heti asennuksen jälkeen. Toisin sanoen Splunkilla ei ole kiinteää skeemaa. Sen sijaan se suorittaa kenttien erottelun hakujen aikana. Monet lokimuodot tunnistetaan automaattisesti, kaikki muut voidaan määrittää konfigurointitiedostoissa tai suoraan hakukyselyssä. [19.]

Puutteet ja heikkoudet

Splunk ei kuitenkaan tarjoa koko rajoittamatonta versiotaan ilmaiseksi eikä sen lähdekoodi ole avoin. Sen ilmainen versio rajoittaa päivittäistä indeksoitavan datan määrää, jolloin dataa on mahdollista indeksoida 500 MB päivässä. Maksullisessa versiossa hinta määräytyy indeksoitavan datan määrän mukaan, joten isojen datamäärien tallentaminen voi käydä kalliiksi. Toisena heikkoutena voidaan tämän vertailun yhteydessä ajatella sen käyttötarkoituksen monipuolisuuden puutetta, sillä se ei sovellu oikeastaan mihinkään muuhun kuin laitteiden tuottaman datan hakuun ja analysointiin. Muita merkittäviä heikkouksia Splunkilla ei ole.

Suosio ja asiakasyritykset

Splunkia käyttää muun muassa Vodafone, Nasdaq, Ubisoft, Valve, Coca Cola sekä monet yliopistot ympäri maailmaa. Esimerkiksi Coca Cola käyttää Splunkia datan analysointiin ja siten asiakaskokemuksien parantamiseen. Valve käyttää Splunkia liiketoiminnan kasvun tukena. [20.]

Vertailu muihin hakualustoihin

Splunkia on kokonaisuutena hankala verrata Apache Solriin, koska niiden käyttötarkoitukset ovat erilaiset. Molemmat kuitenkin ovat nopeita hakualustoja ja oman käyttötarkoituksensa huipputuotteita. Splunkia voidaan kuitenkin verrata Elasticsearchiin, joka toimii yhdessä Logstash-komponentin kanssa, joka ajaa saman asian kuin Splunk. Menemättä syvemmälle lokitiedostojen käsittelyyn todettakoon, että Elasticsearch voi olla parempi vaihtoehto Splunkille, jos sen muita ominaisuuksia halutaan hyödyntää ja sen lisäksi Elasticsearch on myös ilmainen.

6.3 Sphinx

Mikä on Sphinx?

Sphinx on avoimen lähdekoodin hakualusta, joka tarjoaa tekstihakuominaisuuksia asiakassovelluksiin. Sphinxia voidaan käyttää joko itsenäisenä palvelimena tai tietokantamoottorina MySQL-tietokannoille. Sphinx on kirjoitettu C++ -ohjelmointikielellä, ja se toimii Linux-, (RedHat, Ubuntu jne.), Windows-, MacOS-, Solaris-, FreeBSD- ja muutamalla muulla käyttöjärjestelmällä. Sphinxin avulla voidaan joko jakaa indeksi- ja hakutietoja SQL-tietokannasta, NoSQL-tallennustilasta tai tiedostoista nopeasti ja helposti tai indeksoida ja etsiä dataa lennossa työskentelemällä Sphinxin kanssa ikään kuin tietokantapalvelimen kanssa. Sphinx tukee Java-, PHP-, Python-, Perl-, C- ja muutamaa muuta ohjelmointikieltä. [21.]

Käyttötarkoitus, ominaisuudet ja vahvuudet

Hakujen suorittaminen SphinxAPI:n kautta on yksinkertaista. Myös kyselyiden luominen on helppoa, ja hakulausekkeet ilmaistaan SQL:llä. Kun suorittaa SQL-kyselyn Sphinxin kautta, se hakee tiedot tietokannasta ja muodostaa käänteisen indeksin, joka Sphinxissä on kuin hashtable (hajautustaulu), jossa avain on 32-bittinen kokonaisluku, joka lasketaan crc32-algoritmin avulla hakusanasta ja arvo on lista asiakirjojen id:itä, joissa hakusana esiintyy. Sphinxin käyttöönotto on helppoa ja sillä on hyvä dokumentaatio. Sphinx on myös hyvä merkityksellisyyden arvioinnissa ja tarjoaakin useita eri algoritmeja sitä varten. Sphinx tukee hakuja useilla eri kielillä. Sphinxissä on myös mahdollisuus luoda omia merkityksellisyyttä arvioivia algoritmeja tai muokata Sphinxin tarjoamia algoritmeja sopivaksi omaan tarkoitukseen. Sphinx on erityisen hyvä tietokannoista suoraan hakemisessa. Se toimii hyvin yhteen MySQL-tietokantojen kanssa, ja sen indeksointi on erittäin nopea.

Puutteet ja heikkoudet

Sphinxillä on kuitenkin muutamia merkittäviä rajoituksia, joissa se ei pärjää kilpailijoilleen. Sphinx on keskittynyt nimenomaan tietokannasta suoraan hakemiseen, eikä se kykene esimerkiksi indeksoimaan dataa suoraan CSV-, PDF- tai WORD-tiedostoina kuten esimerkiksi Apache Solr pystyy. Dataa ei siis pysty myöskään hakemaan suoraan näissä formaateissa. Sphinx ei toimi erityisen hyvin sovelluksissa,

joissa käytetään JSON- tai XML-formaattia hakutuloksien saamisessa. Sphinx ei myöskään sisällä sisäänrakennettua replikointiominaisuutta, mutta se replikointi on kuitenkin mahdollista. Sphinx on yleisesti ottaen ilmainen, mutta tietyissä käyttötarkoituksissa sen käytöstä joutuu maksamaan.

Suosio ja asiakasyritykset

Sphinx ei ole yhtä suosittu kuin Apache Solr, Elasticsearch tai Splunk, mutta sillä on kuitenkin muutamia isoja yrityksiä asiakkaanaan. Tunnetuin näistä on Craigslist, joka ajaa Sphinxin kautta yli 300 miljoonaa hakua päivittäin. Muita asiakkaita ovat esimerkiksi Avito (Venäjän isoin ja maailman kolmanneksi isoin työpaikkailmoitussivusto), Youku (Kiinan suurin videontoistopalvelu), Tumblr ja The Pirate Bay. [22.]

Vertailu muihin hakualustoihin

Apache Solriin verrattuna Sphinx ei vaikuta kovinkaan hyvältä vaihtoehdolta. Solr tarjoaa paljon enemmän ominaisuuksia, on paljon skaalautuvampi ja kykenee hakemaan dataa kokonaisina asiakirjoina, jotka voivat sisältää dataa lähes missä tahansa formaatissa. Solr on kirjoitettu Java-ohjelmointikielellä, kun taas Sphinx on kirjoitettu C++:lla. Sphinx saattaa siis olla helpompi yhdistää C++:lla ohjelmoituun sovellukseen kuin Solr ja Solr helpompi Java-pohjaiseen sovellukseen. Sphinx ei kuitenkaan vaadi yhtä paljon konfigurointia kuin Solr, jotta sen saa toimimaan sovelluksen kanssa. Sphinxillä on tunnetusti hyvät viralliset dokumentaatiot, mutta Apache Solrin yhteisö on paljon suurempi, mikä voi hyödyttää virhetilanteiden ratkaisemisessa enemmän. Valitsisin Apache Solrin tai Elasticsearchin mieluummin kuin Sphinxin lähes missä tahansa tilanteessa. Ainoastaan jos sovellus, jossa hakualustaa tarvitaan, olisi C++:lla ohjelmoitu ja dataa haettaisiin pääosin suoraan tietokannasta, voisin harkita Sphinxin käyttöä, mutta silti miettisin, voisiko saman tehdä lähes yhtä helposti tai jopa helpommin Apache Solrilla tai Elasticsearchillä.

6.4 Algolia

Mikä on Algolia?

Algolia on suhteellisen uusi hakualusta verrattuna muihin tässä työssä läpikäytyihin hakualustoihin. Se on perustettu vuonna 2012, kun taas suurin osa aikaisemmin työssä mainituista on perustettu 2000-luvun alkupuolella. Algolia eroaa aikaisemmin mainituista hakualustoista myös siten, että se tarjoaa hakualustansa SaaS (Software as a service) -mallin kautta. SaaS tarkoittaa ohjelmiston hankkimista verkkopalveluna perinteisen lisenssipohjaisen tavan ja ohjelmiston asentamisen sijaan. Algolia ei perustu perinteiseen Apache Lucene-hakukoneohjelmistokirjastoon kuten Solr ja Elasticsearch. Algolia on rakennettu nimenomaan puolijäsennettyjen tietojen hakuun käyttäjäkohtaisen haun tehostamiseksi. Algolia ei ole avoimen lähdekoodin tuote eikä se tarjoa palveluaan ilmaiseksi. Algolian ilmainen versio rajoittaa hakujen määrää, tallennettavien asiakirjojen kokoa ja synonyymien määrää eikä se sisällä kaikkia Algolian ominaisuuksia. Algolia on kirjoitettu C++-ohjelmointikielellä ja sen API tukee Java-, PHP-, Python-, Ruby-, C#-, JavaScript- ja Scala-ohjelmointikieliä. Algolia tarjoaa API:nsa myös iOS- ja Android-alustoille. [23.]

Käyttötarkoitus, ominaisuudet ja vahvuudet

Algolia keskittyy erityisesti nopeuteen, kirjoitusvirhesietoisuuteen ja merkityksellisyyteen. Näissä kaikissa se saattaa olla jopa paras tässä työssä verrattavista hakualustoista. Algolia on nopeudeltaan keskimäärin jopa 10-20 kertaa nopeampi kuin Elasticsearch. Algolian reaaliaikainen haku on nopeudeltaan huippuluokkaa, ja siten se pystyy ehdottamaan käyttäjille hakutuloksia käytännössä reaaliajassa loppukäyttäjän kirjoittaessaan kirjain kerrallaan hakukyselyään. Algolian kirjoitusvirhesietoisuus on hyvä ja myös helposti muokattavissa. Algolian merkityksellisyyttä arvioiva algoritmi on suunniteltu jäljittelemään tapaa, jolla ihmisen aivot luonnollisesti etsivät informaatiota. Algolian indeksoinnin voi tehdä muutamalla eri tavalla, mutta suositeltu tapa on tuoda olemassa olevat tiedot tietokannasta Algoliaan sen API:n avulla. Algolia tukee myös indeksien replikointia. Algolia loistaa juurikin sen nopeudessa, kirjoitusvirhesietoisuudessa sekä merkityksellisyyden arvioinnissa. Se tukee kuitenkin paljon muitakin hyödyllisiä ominaisuuksia kuten luokittelua (faceting), synonyymejä ja paikkatietoja. Algolia tukee kaikkia kieliä mukaan lukien

symbolipohjaiset kielet kiina, korea ja japani. Algolia on myös helppo ottaa käyttöön, ja se toimii hyvin oletusasetuksineen ilman erillistä konfigurointia. [24.]

Puutteet ja heikkoudet

Algolian suurin heikkous on sen maksullisuus. Monet yritykset valitsevat mieluummin ilmaisen hakualustan, joka sisältää samat ominaisuudet, vaikka se ei olisikaan yhtä nopea ja sen käyttöönotto olisi vaikeampaa. Algolia ei myöskään tue sen muokkausta, sillä sen lähdekoodi ei ole avointa.

Suosio ja asiakasyritykset

Huolimatta Algolian maksullisuudesta se on saavuttanut tuhansia asiakkaita ja sitä voidaan pitää suosittuna hakualustana, vaikka sen suosio ei olekaan lähelläkään ilmaisia Apache Solria tai Elasticsearchiä. Algolian suurimpiin asiakkaisiin kuuluvat muun muassa Twitch, Lacoste, Quicksilver, Strava, Docker ja Periscope. [25.]

Vertailu muihin hakualustoihin

Ominaisuuksiensa puolesta Algoliaa voidaan pitää osittain jopa parempana kuin Apache Solria tai Elasticsearchiä. Ainakin se on nopeampi ja se sisältää paremman kirjoitusvirhesietoisuuden. Algolia tarjoaa hyvän toimivan kokonaisuuden, mutta jos se ei riitä, ei sitä itse pysty muokkaamaan tai siihen pysty lisäämään omia muutoksiaan. Tällaisissa tapauksissa Apache Solr ja Elasticsearch ovat parempia vaihtoehtoja, koska niiden lähdekoodi on avointa ja niitä pystyy muokkaamaan tarpeiden mukaisiksi. Algolia on myös helppokäyttöisempi ja helpommin käyttöönotettava hakualusta kuin Apache Solr ja Elasticsearch. Monella pienellä yrityksellä kuitenkin raha ratkaisee ja siksi hakualustan valinta ei kohdistu Algoliaan, joka ei hintansa takia välttämättä ole paras hakuratkaisu pienelle yritykselle.

6.5 Amazon CloudSearch

Mikä on Amazon CloudSearch?

Vuonna 2012 Amazon julkaisi oman hakualustansa Amazon CloudSearchin. Amazon CloudSearchin käytössä hyödynnetään Amazon Web Services (AWS) -palvelua hakualustan ylläpidossa. Toisin sanoen CloudSearch pyörii Amazonin tarjoamilla palvelimilla. Tämä on hyödyllistä silloin, kun yritys ei halua ylläpitää omia palvelimiaan. AWS-palvelu on erittäin suosittu, mutta se on myös maksullinen ja niin on myös Amazon CloudSearch. Valitsin Amazon CloudSearchin vertailuun, koska myös tämän työn tilanneessa yrityksessä on mietitty siirtymistä AWS:n käyttöön ainakin DCS-projektin osalta. Jos vaihto toteutettaisiin, voitaisiin mahdollisesti myös hakualusta siirtää AWS:ään (eli kenties vaihtaa Amazon CloudSearchiin), jolloin kaikki DCS:n tarjoamat palvelut pyörisivät AWS:ssä, eikä omia palvelimia tarvitsisi ylläpitää. Amazon CloudSearch on hallinnoitu palvelu AWS Cloudissa, mikä tekee siitä yksinkertaisen ja kustannustehokkaan ottaa käyttöön, hallita ja skaalata verkkosivuston tai -sovelluksen hakuratkaisu. Kuten Algolia, myös Amazon CloudSearch tarjoaa hakualustansa SaaS (Software as a service) -mallin kautta. Amazon CloudSearch tukee 34:ää kieltä ja suosittuja hakutoimintoja, kuten korostusta (highlighting), automaattista täydennystä ja paikkatietojen hakua. CloudSearch tarjoaa ohjelmointirajapintansa Java-, Ruby-, Python-, .Net-, PHP- ja Node.js -ohjelmointikielillä. CloudSearch tukee hakutuloksien palautusta JSON- ja XML-formaateissa. Indeksointi ei onnistu suoraan mistä tahansa tiedostomuodosta kuten PDF:stä niin kuin esimerkiksi Apache Solrissa. [26.]

Käyttötarkoitus, ominaisuudet ja vahvuudet

Amazon CloudSearchin hienous on sen helppokäyttöisyys ja vaivattomuus. Monet sen ominaisuudet on automatisoitu niin, ettei asiakkaan tarvitse huolehtia niistä. Esimerkiksi replikoinnista on turha huolehtia, sillä Amazon CloudSearch tarjoaa tehokkaan autoskaalauksen. Eli kun datan tai kyselyjen määrä vaihtelee, Amazon CloudSearch voi skaalata hakujen resursseja ylös tai alas tarpeen mukaan. Toisin sanoen hakujen lisääntyessä se voi vaihtaa palvelimen toiselle AWS-palvelimelle, joka kykenee käsittelemään enemmän dataa ja kyselyitä, tai se voi replikoida eli monistaa itsensä useammalle palvelimelle. Skaalausta voi hallita myös itse esimerkiksi, jos tietää, että hakuliikenne tulee lähitulevaisuudessa lisääntymään ja tarvitaan lisää kapasiteettia. Amazon CloudSearch on täysin hallittu mukautettu hakupalvelu. Laitteisto- ja

ohjelmistopäivitykset, asennus ja konfigurointi, ohjelmistojen korjaus, datan osiointi eli partitiointi, palvelimien seuranta, skaalaus ja datan kestävyys käsitellään automaattisesti. CloudSearch käyttää Apache Solria ja siten se pystyy tarjoamaan useita suosittuja hakualustaominaisuuksia, jotka ovat käytettävissä Apache Solr -palvelun kanssa.

Puutteet ja heikkoudet

Amazon CloudSearchin suurimmat heikkoudet ovat sen maksullisuus ja se, ettei sen lähdekoodi ole avointa eli sitä ei juurikaan pysty muokkaamaan. CloudSearch ei kuitenkaan ole erittäin kallis, sillä sen hinta perustuu hakujen ja datan määrään eikä esimerkiksi kiinteään kuukausihintaan riippumatta siitä, paljonko dataa indeksoidaan ja paljonko hakuja toteutetaan.

Suosio ja asiakasyritykset

Voisi kuvitella, että Amazon CloudSearch on suosittu hakualusta, koska se on Amazonin tarjoama ja se sisältää lähes kaikki ominaisuudet, mitä isoimmat hakualustat tarjoavat. CloudSearch ei kuitenkaan ole erityisen suosittu isojen ja tunnettujen yritysten keskuudessa. Amazon CloudSearchiä käyttävät esimerkiksi yritykset kuten Bizo, News UK, Company Check, PBS ja SmugMug. [27.]

Vertailu muihin hakualustoihin

Amazon CloudSearchin ominaisuudet perustuvat Apache Solriin, joten näiden kahden hakualustan välillä ominaisuudet ovat kutakuinkin samat. Suurin ero on kuitenkin se, että Apache Solr on ilmainen ja sen lähdekoodi on täysin avointa. Tämän ansiosta Solr on täysin muokattavissa omien tarpeiden mukaiseksi, kun taas CloudSearch ei. Toisaalta CloudSearch on tehty tarkoituksella valmiiksi paketiksi, jonka käyttöönotto on erittäin helppoa ja sen ylläpito tapahtuu automaattisesti. Apache Solrissa mikään ei ole automaattista vaan kaikki tehdään ja konfiguroidaan itse sellaiseksi kuin ne halutaan ja myös replikointi ja muu ylläpito hoidetaan itse. Apache Solr pystyy myös indeksoimaan Word- ja PDF-asiakirjoja toisin kuin Amazon CloudSearch. Yleisesti ottaen Amazon CloudSearch vaikuttaa hyvältä vaihtoehdolta, joka on erittäin helppo ottaa käyttöön eikä sen ylläpito aiheuta päänvaivaa. Vaikka Amazon CloudSearch onkin maksullinen, sen helppous säästää paljon aikaa ja työtunteja eli siinä mielessä myös rahaa. On kuitenkin

vaikea arvioida, miksi Amazon CloudSearch ei ole suositumpi isojen yrityksien keskuudessa. Esimerkiksi Algolia-hakualusta omaa samat rajoitteet (maksullisuus ja suljettu lähdekoodi), mutta silti sillä on isoja yrityksiä asiakkainaan.

7 Vertailun yhteenveto ja toimenpide-ehdotukset

Tässä luvussa käydään läpi hakualustojen vertailun yhteenveto ja vedetään niistä johtopäätöksiä. Luvussa kerrotaan, mitä asioita tulee ottaa huomioon hakualustaa valittaessa ja mitä niistä kannattaa painottaa eniten. Lopuksi esitän toimenpide-ehdotukseni tämän työn tilanneelle yritykselle, joka käytti DCS-projektissaan työntekohetkellä Apache Solria hakualustanaan.

7.1 Vertailun yhteenveto

Miten valita hakualusta?

Luvussa 6 tehdyn vertailun pohjalta huomataan, että hakualustoilla on paljon eroja ja osa niistä on luotu hieman eri tarkoitukseen kuin toiset, eikä ole yhtä tiettyä käyttötarkoitusta, johon ne kaikki sopisivat hyvin. Esimerkiksi Splunk-hakualusta on luotu nimenomaan lokitiedostojen käsittelyyn. Osa hakualustoista on täysin ilmaisia ja muokattavissa olevia ja osa on maksullisia, eikä niitä pysty juurikaan muokkaamaan.

Ei voida sanoa yksinkertaisesti, mikä hakualustoista on paras. Hakualustan valitseminen tulee aina perustua käyttötarkoitukseen. Tällöin paras hakualusta on se, joka sopii parhaiten omaan käyttötarkoitukseen. Toki yrityksen on syytä myös miettiä, onko se valmis maksamaan esimerkiksi Algolian tai Amazon CloudSearchin hinnan, jotta hakualustan käyttöönotto ja ylläpito olisivat helpompia vai kannattaako sen käyttää enemmän työvoimaa ja -tunteja muuten ilmaisen hakualusta käyttöönottoon ja ylläpitoon. Joskus yritys voi tarvita avoimen lähdekoodin tarjoamaa muokattavuutta omien ominaisuuksien toteuttamiseen hakualustaan. Tällöin parhaita vaihtoehtoja käyttötarkoituksesta riippuen on yleensä Elasticsearch tai Apache Solr. Käyttötarkoituksen ja hinnan lisäksi tärkeä ominaisuus hakualustan valinnassa on myös sen tarjoamat kielet, joilla hakuja voidaan suorittaa. Jos yrityksen projekti, jossa hakualustaa tarvitaan, halutaan myöhemmin esimerkiksi laajentaa useammille kielille,

on syytä ennen valintaa jo miettiä, mitä kieliä mahdollisesti halutaan tukea ja joudutaanko joku hakualustavaihtoehdoista hylkäämään siksi, ettei se tue jotakin kieltä.

Jakaisin vertailun hakualustat valitsemista helpottaakseni kahteen pääryhmään: lähes mitä tahansa dataa käsittelevät hakualustat ja lokitiedostoja käsittelevät hakualustat. Nämä jakaisin vielä ilmaisiin ja maksullisiin hakualustoihin. Elasticsearch kykenee hyvin käsittelemään myös lokitiedostoja sen Logstash-komponentin ansiosta, joten asettaisin sen siksi myös siihen kategoriaan. Tämä jako ja sitä havainnollistava taulukko 3 auttavat hakualustan valinnassa varteenotettavien vaihtoehtojen eristämisessä muista.

Taulukko 3. Vertailun hakualustat jaettuna kategorioihin.

Lähes mitä tahansa käsittelevät hakualustat		**Lokitietoja käsittelevät hakualustat**	
Ilmaiset	**Maksulliset**	**Ilmaiset**	**Maksulliset**
Elasticsearch	Algolia	Elasticsearch	Splunk
Apache Solr	Amazon CloudSearch		
Sphinx			

Jos oletetaan, että hakualustan valitseminen on niinkin helppoa, että valitaan vain taulukon 3 sarakkeista sopivin kategoria yrityksen käyttötarkoituksen ja budjetin mukaan, niin vaihtoehtoja jää enää maksimissaan kolme. Tällöin on helpompaa, kun tarvitsee verrata vain muutamaa hakualustaa keskenään.

Johtopäätöksiä

Lokitiedostoja käsittelevistä hakualustoista Splunk on hyvä valinta, jos käyttötarve on nimenomaan vain lokitiedostojen tai muun laitedatan hyödyntäminen. Elasticsearch tarjoaa tämän päälle paljon muitakin ominaisuuksia, mutta ei ehkä ole paras vaihtoehto vain tähän käyttötarkoitukseen ominaisuuksien puolesta, sillä sitä ei ole alun perin suunniteltu sitä varten. Elasticsearch on kuitenkin erittäin varteenotettava vaihtoehto, ja se on suosittu myös lokitiedostojen hallinnassa. Splunk on myös maksullinen, kun taas Elasticsearch on ilmainen, mikä selittää, miksi Elasticsearch on kerännyt paljon suosiota myös tältä osa-alueelta. Lokitiedostojen käsittelyyn löytyy kuitenkin paljon muitakin vaihtoehtoisia työkaluja, mutta niihin perehtyminen ei sisälly tähän työhön.

Ilmaisista lähes mitä tahansa dataa käsittelevistä hakualustoista Sphinx erottuu joukosta, koska se ei perustu Apache Luceneen ja siten siitä myös puuttuu ominaisuuksia, jotka Apache Solr ja Elasticsearch tarjoavat. Jos kuitenkin tietää

pärjäävänsä Sphinxin tarjoamilla ominaisuuksilla, on se hieman helpompi konfiguroida ja ottaa käyttöön kuin nämä kaksi muuta. Sphinxin yksi suurimmista eduista Solriin ja Elasticsearchiin verrattaessa on sen tarjoama mahdollisuus luoda omia merkityksellisyyttä arvioivia algoritmeja tai muokata Sphinxin tarjoamia valmiita algoritmeja sopiviksi omiin tarkoituksiin. Solrissa ja Elasticsearchissä merkityksellisyysalgoritmeja on hieman hankala muokata. Sphinx on erityisen hyvä tietokannoista suoraan hakemisessa ja toimii erityisen hyvin yhteen MySQL-tietokantojen kanssa. Myös Solrissa ja Elasticsearchissä on mahdollista hakea dataa suoraan tietokannoista indeksoitavaksi, mutta niissä sitä ei ole optimoitu yhtä sulavaksi kuin Sphinxissä. Kuten huomataan, Apache Solr ja Elasticsearch ovat hyvin samanlaisia Lucenen takia, mutta niillä on kuitenkin pieniä eroja. Elasticsearch tarjoaa datan käsittelyn ainoastaan JSON-muodossa, kun taas Apache Solr tarjoaa sen JSON:in lisäksi myös esimerkiksi XML-muodossa. Elasticsearchin takana on Elastic-niminen yritys, joka työskentelee aktiivisesti Elasticsearchin kanssa ja laajentaa ja parantaa sitä jatkuvasti. Apache Solrilla ei ole virallisia työntekijöitä vaan sen kehitys on täysin yhteisön käsissä. Elasticsearchin kehitykseen voi myös osallistua kuka tahansa, mutta päätöksen siitä, mitä ohjelmoidaan ja mihin suuntaan tuotetta viedään, tekee aina Elastic-yritys.

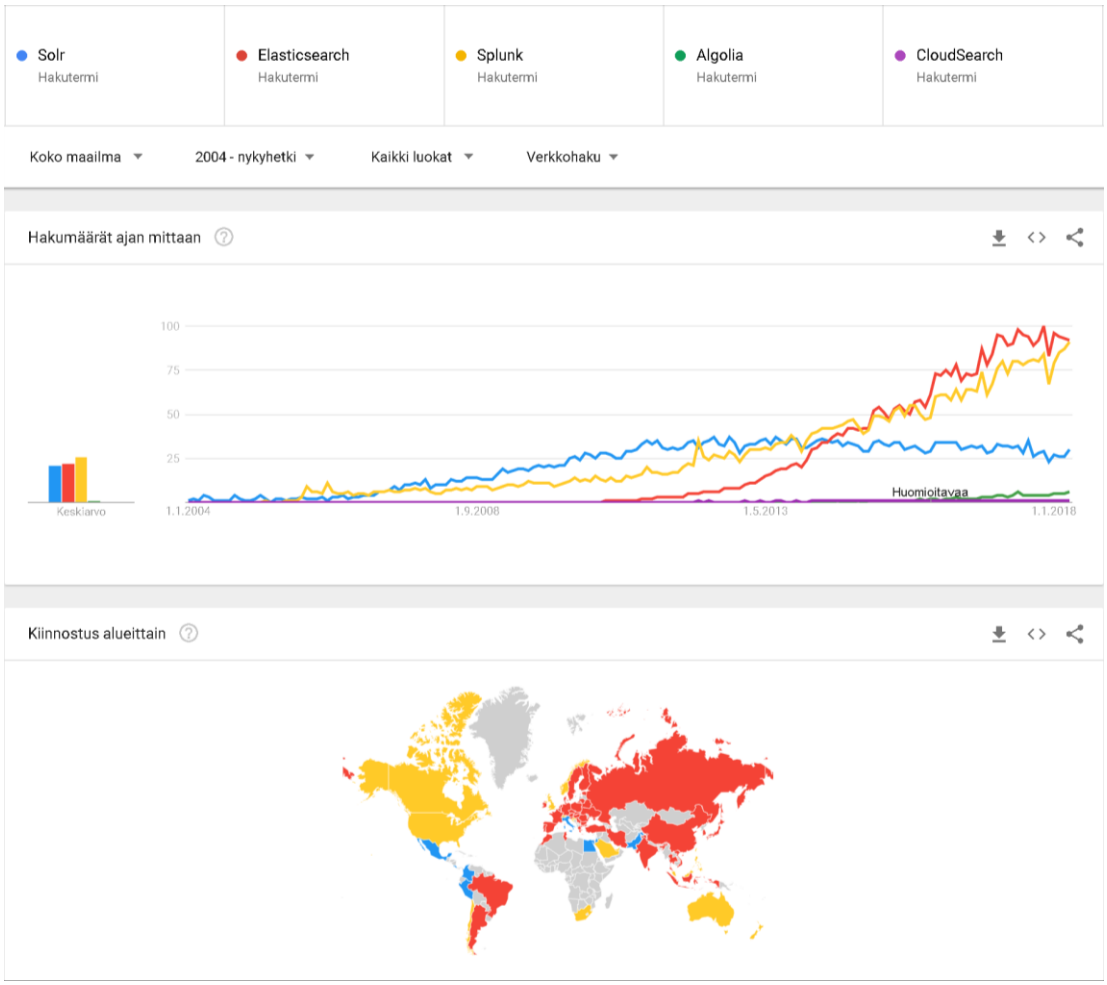
Maksullisia lähes mitä tahansa dataa käsitteleviä hakualustoja vertailussa oli mukana Algolia ja Amazon CloudSearch. Molemmat ovat erittäin helppoja ottaa käyttöön, mutta niiden lähdekoodit ovat myös suljettuja eli niiden muokattavuus on hyvin rajoitettua. CloudSearchissä on painotettu ennen kaikkea helppokäyttöisyyttä ja ylläpidon vaivattomuutta, kun taas Algolia keskittyy nopeuteen ja suorituskykyyn. Iso ero näiden välillä on se, että CloudSearch perustuu Apache Solriin ja sisältää siten Lucenen ominaisuudet ilman Apache Solrin tarjoamaa muokattavuutta. Algoliassa ominaisuudet on toteutettu ilman Lucenen hyödyntämistä. Tämä ei kuitenkaan ole huono asia Algolian tapauksessa, sillä useat testitulokset osoittavat Algolian olevan erittäin nopea. Algolia on keskimäärin noin 10-20 kertaa nopeampi hauissa kuin Elasticsearch, jonka hakuominaisuus perustuu Luceneen kuten myös Solrin ja siten myös CloudSearchin.

Otin vertailuun mukaan Amazonin CloudSearchin, koska myös tämän työn tilanneessa yrityksessä on harkittu siirtymistä AWS:n käyttöön ainakin DCS-projektin osalta ja CloudSearch pyörii nimenomaan Amazonin AWS:ssä. Muista hakualustoista ainakin Elasticsearchiä on mahdollista ajaa AWS:ssä tai muissa vastaavassa pilvipalvelualustoissa.

Kuten vertailussa todettiin, Apache Solr ja Elasticsearch ovat hyvin samanlaisia, koska ne molemmat perustuvat Apache Lucene -hakukoneohjelmistokirjastoon. Nämä kaksi ovat suosituimmat ja selkeästi ainakin tarjoamiensa ominaisuuksien kannalta parhaimmat ilmaiset hakualustat. Elasticsearch on viime vuosina laajentunut nopeasti ja kerännyt paljon suosiota ja onkin nykyään suositumpi kuin Apache Solr, jolla oli hallussaan vielä 2013 suosituimman hakualustan titteli. Kyse ei ole siitä, että Apache Solrin suosio olisi laskenut, sillä se on pysynyt aika lailla samoissa lukemissa. Elasticsearch vain on onnistunut keräämään enemmän uusia asiakkaita. Tämä saattaa osittain johtua siitä, että Elasticsearch on viime vuosien aikana laajentunut huomattavasti enemmän kuin Solr. Elasticsearchiä kutsutaan nykyään myös nimellä Elastic stack, sillä se koostuu Elasticsearch-, Logstash-, Kibana- ja Beats-komponenteista, jotka toimivat yhdessä. Nämä komponentit ovat osa Elasticsearchin laajentumista ja osasyy sen suosion kasvuun.

Suosioiden vertailu

Kuvassa 10 on havainnollistettu hakualustojen suosiota Google-hakujen perusteella. Tämä ei ole paras mahdollinen tapa verrata niiden suosiota, mutta se kelpaa suuntaa antavana tilastona. Vertailussa mukana ollutta Sphinx-hakualustaa ei kyseiseen Google Trends -vertailuun voinut ottaa mukaan, sillä Sphinx-hakusanaa käytetään muussakin yhteydessä kuin vain Sphinx-hakualustaan liittyen. Kuvasta voidaan kuitenkin havaita iso ero suosiossa maksullisten Algolian ja Amazonin CloudSearchin ja ilmaisten hakualustojen välillä. Splunkin suuri suosio johtuu sen tarjoamasta erittäin hyvästä lokitiedostojen käsittelystä. Tässä työssä vertailussa olleista hakualustoista vain Splunk on suunniteltu tähän käyttötarkoitukseen, mutta myös Elasticsearch kykenee siihen Logstash-komponentin avulla. Kuvan kartasta havaitaan missä päin maailmaa mikäkin hakualusta on suosituin. Kun Splunk poistetaan yhtälöstä, kaikki keltaiset alueet muuttuvat punaisiksi ja siten lähes koko kartta on punainen, joka tarkoittaa sitä, että Elasticsearch on suosituin hakualusta lähes kaikkialla maailmassa.



Kuva 10. Google Trendsin tarjoama vertailu Apache Solr-, Elasticsearch-, Splunk-, Algolia- ja Amazon CloudSearch -hakukoneiden suosiosta Google-hakujen perusteella [28].

Taulukossa 4 on havainnollistettu Algolian työntekijöiden suorittaman nopeustestin tulokset, jossa luotiin yksi Algolia-toteutus yhdellä shardilla, ja toinen viidellä shardilla ja samoin Elasticsearch-toteutukset yhdellä ja viidellä shardilla. Kaikissa tapauksissa indeksoituna datana toimi IMDB-sivuston tarjoama data, joka koostui 400 tuhannesta näyttelijästä ja kahdesta miljoonasta elokuvasta ja tv-sarjasta. Tuloksien perusteella Algolia on huomattavasti nopeampi kuin Lucene-pohjainen Elasticsearch. Tällaisissa testeissä tulee kuitenkin huomioida, että kyse on vain nopeudesta eikä siinä ole huomioitu sitä, olivatko hakutulokset merkityksellisyyden puolesta hyviä. Testi saattaa olla myös jossain määrin puolueellinen, sillä sen on tehnyt Algolian työntekijät ja sen on julkaissut Algolian CTO eli teknologiajohtaja.

Nopeuksien vertailu

Taulukko 4. Nopeustestin tulokset Algolian ja Elasticsearchin (ES) käytöstä IMDB-sivuston tarjoamasta datasta, joka koostuu elokuvista, tv-sarjoista ja näyttelijöistä [29].

Kysely	**Algolia 1 Shard**	**Algolia 5 Shardia**	**ES 1 Shard**	**ES 5 Shardia**
geo	< 1ms	2ms	101ms	36ms
george clo	2ms	3ms	121ms	51ms
b	< 1ms	2ms	147ms	60ms
batman	2ms	2ms	94ms	29ms
world w	5ms	2ms	134ms	68ms
e	< 1ms	2ms	202ms	81ms
emilia	< 1ms	2ms	102ms	31ms
alexandre b	18ms	9ms	243ms	109ms

Taulukon 4 nopeustesti vertaa Algolian nopeutta Lucene-pohjaisiin hakualustoihin. Tässä työssä käsiteltyjä Lucene-pohjaisia hakualustoja ovat siis Apache Solr, Elasticsearch ja Amazon CloudSearch. Nämä kolme ovat nopeudeltaan siis hyvin lähellä toisiaan.

7.2 Toimenpide-ehdotukset

Lähtökohtana oli, että yritys on tyytyväinen Apache Solrin tarjoamiin ominaisuuksiin ja se on toistaiseksi erittäin riittävä hakuratkaisu yrityksen DCS-projektissa. Jos yritys ei ehdottomasti tahdo vaihtaa helpommin ylläpidettävään, mutta maksulliseen hakualustaan, en suosittele hakualustan vaihtoa DCS-projektissa ainakaan tämän työn perusteella. Vaihtaminen esimerkiksi Elasticsearchiin ei toisi DCS-projektin kannalta lisää hyödyllisiä ominaisuuksia. Vaihtoprosessi on sen verran työläs, ettei vaihtaminen ole järkevää, ellei uusi hakualusta tuo tärkeitä lisäominaisuuksia, jotka vanhasta hakualustasta puuttuvat. DCS-projektissa dataa käsitellään XML-formaatissa, joka ei myöskään ole mahdollista Elasticsearchissä, joten datan käsittely tulisi hakualustan vaihdon yhteydessä vaihtaa silloin JSON-muotoon.

Tulevissa projekteissa yrityksen kannattaa kuitenkin miettiä, onko se valmis maksamaan hakualustasta, jotta sen käyttöönotto ja varsinkin ylläpito helpottuisi. Jos yritys näkee, ettei hyötysuhde ole tarpeeksi suuri, suosittelen Elasticsearchin harkitsemista hakualustaksi, jos projektissa ei ole rajoitteita, jotka sulkisivat Elasticsearchin pois vaihtoehdoista. Elasticsearch ei ole syyttä suosituin hakualusta vuonna 2018. Toisaalta yrityksellä on jo hyviä kokemusta Apache Solrista, ja se on ollut hyvä vaihtoehto yrityksen

tarpeisiin. Solr on edelleen arvostettu ja myös suosittu hakualusta, ja se on edelleen varteenotettava kilpailija Elasticsearchille.

Ei ole olemassa yleistä vastausta siihen, mikä on paras hakualusta. Se riippuu aina käyttötarkoituksesta. Jokaisessa projektissa tulee erikseen miettiä, mikä on sopivin hakualusta kyseiseen tarpeeseen.

8 Yhteenveto

Insinöörityössä perehdyttiin erityisesti Apache Solr -hakualustaan ja sen ominaisuuksiin ja toimintoihin. Tavoitteena oli vertailla Solria muihin vastaaviin hakualustoihin ja selvittää, mikä niistä olisi sopivin työn tilanneen yrityksen tarkoitukseen. Solria tuli verrata muutamaan yleisimpään hakualustaan ja selvittää niiden merkittävimmät eroavaisuudet. Hakualustojen vertailun lisäksi insinöörityön tavoitteena oli myös päivittää käytössä oleva Apache Solr uusimpaan versioon.

Työn tuloksena oli Apache Solr päivitettynä uusimpaan versioon yrityksen sovellukseen sekä hakualustojen vertailu tuloksineen ja pohdintoineen. Annoin myös toimintaehdotukseni yritykselle siitä, tulisiko sen vaihtaa hakualustaa ja miten hakualustan valitseminen kannattaa suorittaa tulevissa projekteissa. Vertailussa havaittiin, että Elasticsearchistä on tullut johtava hakualusta, joka laajenee nopeaa tahtia. Ehdotinkin, että yritys harkitsisi tulevaisuuden projekteissa, onko Elasticsearch hyvä vaihtoehto kyseiseen projektiin. En suositellut yritykselle tällä hetkellä käytössä olevan Apache Solrin vaihtamista tässä vaiheessa, sillä se on riittävä sovelluksen tarkoitukseen ja vaihtoprosessi veisi liikaa aikaa hyötyihin nähden.

Kaikki tämän työn tavoitteet saavutettiin ja vertailussa selvisi mielenkiintoisia asioita. Tämän työn avulla yrityksen ei tarvitse etsiä vaihtoehtoisia hakualustoja Apache Solrille, sillä tässä työssä on käyty läpi kaikki varteenotettavat vaihtoehdot sille. Ilman tätä työtä yritys joutuisi tekemään saman tutkimustyön ja vertailun löytääkseen sopivimman hakualustan tulevissa projekteissa.

Toki työssä ei käyty läpi jokaista olemassa olevaa hakualustaa, mutta niitä ei olekaan kovin montaa. Kaikki suositut hakualustat kuitenkin löytyvät tämän työn vertailusta. Vertailun ulkopuolelle jääneet hakualustat eivät olleet merkittäviä eivätkä suosittuja.

Työ oli kokonaisuudessaan erittäin mielenkiintoinen ja opettava prosessi, sillä minulla ei ollut aikaisempaa kokemusta tässä työssä käsitellyistä aiheista. Hakualustat, Scala-ohjelmointikieli, PostgreSQL ja Java Servlet Containerit (tässä tapauksessa Apache Tomcat) olivat kaikki minulle uutta. Opin kuitenkin paljon ja varsinkin palvelinpuolen (back end) ymmärrykseni kasvoi merkittävästi insinöörityön aikana.

Lähteet

1 Grainger, Trey & Potter, Timothy. 2014. Solr in Action. Yhdysvallat: Manning.

2 Hicks, Matt. 2014. Java vs. Scala: Why Should I Learn Scala? Verkkoaineisto. Toptal. <https://www.toptal.com/scala/why-should-i-learn-scala>. Luettu 21.2.2018.

3 Introduction Scala documentation. Verkkoaineisto. Scala. <http://docs.scala-lang.org/tour/tour-of-scala.html>. Luettu 21.2.2018.

4 Scalatra. Verkkoaineisto. <http://scalatra.org/>. Luettu 22.2.2018.

5 PostgreSQL: About. Verkkoaineisto. <https://www.postgresql.org/about/>. Luettu 22.3.2018.

6 Apache Tomcat. Verkkoaineisto. <http://tomcat.apache.org/>. Luettu 22.2.2018.

7 Ellis, Graham J. 2004. Tomcat Overview. Verkkoaineisto. <http://www.wellho.net/downloads/A651.pdf>. Luettu 22.2.2018.

8 Vukotic, Aleksa & Goodwill, James. 2011. Apache Tomcat 7. Yhdysvallat: Apress.

9 Apache Tomcat Clustering. Verkkoaineisto. <https://www.roguewave.com/resources/white-papers/increase-system-availability-by-leveraging-apache>. Luettu 23.2.2018.

10 Companies Using Lucene/Solr. Lucid Works. Verkkoaineisto. <https://lucidworks.com/2012/01/21/who-uses-lucenesolr/>. Luettu 7.3.2018.

11 Faceted search with Solr. Verkkoaineisto. <https://lucidworks.com/2009/09/02/faceted-search-with-solr/>. Luettu 28.2.2018.

12 Why Faceted Search Is The Most Honest Kind of Search. Verkkoaineisto. <https://webkite.com/2014/faceted-search-fridays-faceted-search/>. Luettu 28.2.2018.

13 Solr Replication Alfresco Documentation. Verkkoaineisto. <https://docs.alfresco.com/5.2/concepts/solr-replication.html>. Luettu 2.3.2018.

14 SolrCloud Apache Solr Reference Guide 7.3. Verkkoaineisto. <https://lucene.apache.org/solr/guide/7_3/solrcloud.html>. Luettu 5.3.2018.

15 Open Source Search and Analytics. Elasticsearch. Elastic. Verkkoaineisto. <https://www.elastic.co/>. Luettu 28.3.2018.

16 DB-Engines Rranking. Popularity ranking of search engines. Verkkoaineisto. <https://db-engines.com/en/ranking/search+engine>. Luettu 28.3.2018.

17 Use Cases. Elastic Stack Success Stories. Elastic. Verkkoaineisto. <https://www.elastic.co/use-cases>. Luettu 28.3.2018.

18 SIEM, AIOps, Application Management, Log Management, Machine Learning and Complience Splunk. Verkkoaineisto. <https://www.splunk.com/>. Luettu 4.4.2018.

19 What Is Splunk And How Does It Work? Helge Klein. Verkkoaineisto. <https://helgeklein.com/blog/2014/09/splunk-work/>. Luettu 4.4.2018.

20 Customers. Splunk. Verkkoaineisto. <https://www.splunk.com/en_us/customers.html>. Luettu 4.4.2018.

21 About. Sphinx. Verkkoaineisto. <http://sphinxsearch.com/about/sphinx/>. Luettu 4.4.2018.

22 Powered By. Sphinx. Verkkoaineisto. <http://sphinxsearch.com/info/powered/>. Luettu 5.4.2018.

23 Algolia. The Most Reliable Platform for Building Search. Verkkoaineisto. <https://www.algolia.com/>. Luettu 5.4.2018.

24 What Is Algolia? Getting Started Guide. Algolia Documentation. Verkkoaineisto. <https://www.algolia.com/doc/guides/getting-started/what-is-algolia/>. Luettu 5.4.2018.

25 Algolia. Customers And Case Studies. Verkkoaineisto. <https://www.algolia.com/customers>. Luettu 5.4.2018.

26 AWS. Amazon CloudSearch. Product Details. Verkkoaineisto. <https://aws.amazon.com/cloudsearch/details/>. Luettu 6.4.2018.

27 AWS. Amazon CloudSearch. Testimonials. Verkkoaineisto. <https://aws.amazon.com/cloudsearch/testimonials/>. Luettu 6.4.2018.

28 Solr, Elasticsearch, Splunk, Algolia, CloudSearch. Tutki. Google Trends. Verkkoaineisto. <https://trends.google.fi/trends/explore?date=all&q=Solr,Elasticsearch,Splunk,Algolia,CloudSearch>. Luettu 9.4.2018.

29 Full Text Search In Your Database. Algolia vs Elasticsearch. Algolia Blog. Verkkoaineisto. <https://blog.algolia.com/full-text-search-in-your-database-algolia-versus-elasticsearch/>. Luettu 11.4.2018.