Teemu Pöytäniemi

Suurten tiedostojen siirto ja talletus eri pilvialustoilla

Diplomityö

Tieto- ja säjkötekniikan tiedekunta

Mika Välimäki

TIIVISTELMÄ

Teemu Pöytäniemi: Suurten tiedostojen siirto ja talletus eri pilvialustoilla

Diplomityö

Tampereen yliopisto

Tietotekniikan tutkinto-ohjelma

2021

Tiivistelmä on suppea, 1 sivun mittainen itsenäinen esitys työstä: mikä oli ongelma, mitä tehtiin ja mitä saatiin tulokseksi. Kuvia, kaavioita ja taulukoita ei käytetä tiivistelmässä.

Laita työn pääkielellä kirjoitettu tiivistelmä ensin ja käännös sen jälkeen. Suomenkieliselle kandidaatintyölle pitää olla myös englanninkielinen nimi arkistointia varten.

Tässä pohjassa tiivistelmää varten 2 omaa tekstityyppiä: tunnistetiedoille tyyli CoverBodyText2 ja tiivistelmätekstille Abstract, jossa riviväli on 1.0. Otsikkotyyppi on Heading (no number), joka tekee automaattisesti sivunvaihdon (Page break before). Samaa otsikkotyyppiä käytetään mm. sisällysluettelossa. Lähdeluettelossa on identtinen tyyppi hieman eri nimellä, jolloin se voidaan poimia sisällysluetteloon. Sivunumeroja varten etusivun lopussa pitää olla Section Break ja tiivistelmän yläotsakkeen (header) asetus Link to Previous pois päältä, ja lisäksi sivunumeron muotoilusta Start at i (eikä Continue).

Avainsanat: Tiivistelmä-tekstin jälkeen.

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin OriginalityCheck –ohjelmalla.

abstract

:

Tampere University

The abstract is a concise 1-page description of the work: what was the problem, what was done, and what are the results. Do not include charts or tables in the abstract.

Put the abstract in the primary language of your thesis first and then the translation (when that is needed).

This document template has two text styles for abstract. BibInfo is for bibliographical information above whereas the rest uses the style Abstract, which has line spacing of 1.0. The style Heading (no number) is used in the frontmatter before actual text and it makes the necessary preceding page break. Similar style is used in the bibliography with slightly different name in order to include it in the table of contents. The title page must end with Section Break to get pages numbered correctly. Moreover, the header on this page turns off the setting Link to Previous and formats the page numbers to Start at 1 (instead of Continue).

Keywords: After Abstract-text

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

ALKUSANAT

Tämä dokumenttipohja on laadittu Tampereen yliopiston tekniikan alan opinnäytetöitä varten. Mallipohja perustuu aikaisemmalla Tampereen teknillisen yliopiston pohjalle, mutta se on päivitetty vuonna 2019 toimintansa aloittavaa Tampereen yliopistoa varten.

Alkusanoissa esitetään opinnäytetyön tekemiseen liittyvät yleiset tiedot. Tapana on myös esittää kiitokset työn tekemiseen vaikuttaneille henkilöille ja yhteisöille. Alkusanat eivät kuulu arvioinnin piriin, mutta niissä ei silti ole sopivaa moittia tai kritisoida ketään. Alkusanojen pituus on enintään 1 sivu. Alkusanojen lopussa on päivämäärä, jonka jälkeen työhön ei ole enää tehty korjauksia.

Tampereella, 15.2.2019

Päivittäjä

Sisällysluettelo

[1. Johdanto 1](#_Toc73893651)

[2. Teoria 2](#_Toc73893652)

[2.1 Tiedostojen siirto 2](#_Toc73893653)

[2.1.1 HTTP ja WebSocket 2](#_Toc73893654)

[2.1.2 FTP ja FTPS 6](#_Toc73893655)

[2.1.3 SFTP 8](#_Toc73893656)

[2.2 Tiedostojen talletus 8](#_Toc73893657)

[2.2.1 SQL tietokanat 9](#_Toc73893658)

[2.2.2 Object Storage 9](#_Toc73893659)

[2.2.3 Block storage 11](#_Toc73893660)

[2.2.4 File storage 12](#_Toc73893661)

[3. esimerkki projektit 14](#_Toc73893662)

[3.1 Tiedostojen siirto 14](#_Toc73893663)

[3.2 Tiedostojen talletus 14](#_Toc73893664)

[4. Projektien tulokset 15](#_Toc73893665)

[4.1 kustannusvertailu 15](#_Toc73893666)

[4.2 tehokkuusvertailu 15](#_Toc73893667)

[5. yhteenveto 16](#_Toc73893668)

[Lähteet 18](#_Toc73893669)

KUVALUETTELO

[**Kuva 1.** Kuvaaja on hyvä muokata julkaisukelpoiseksi. Vasemmalla on esitetty muokkaamaton kuvaaja ja oikealla muokattu. **Error! Bookmark not defined.**](#_Toc532040187)

[**Kuva 2.** Tyylit kirjoitusohjeen versioissa 11.3 ja 11.5 20](#_Toc532040188)

Tämä luettelo on vapaaehtoinen. Kuvaluettelo lisätään *References > insert Table of Figures* ja sieltä *Options… > Build table of figures based on > Style:Figure Caption*. Myös taulukkoluettelon saa samasta kohdasta, kun valitsee viimeisestä kohdasta tyylin *Table Caption*.

LYHENTEET JA MERKINNÄT

CC-lisenssi Creative Commons -lisenssi

LaTeX ladontajärjestelmä tieteelliseen kirjoittamiseen

SI-järjestelmä ransk. Système international d’unités, kansainvälinen mittayksikköjärjestelmä

TTY Tampereen teknillinen yliopisto

URL engl. Uniform Resource Locator, verkkosivun osoite

*a* kiihtyvyys

**F**voima

*m* massa

.

# Johdanto

Moderni pilvilaskentainfraktuuri sai alkunsa 1990-luvulla, kun ensimmäiset yritykset alkoivat tarjota virtuaalikoneita ja virtuuliverkkoja SaaS(software-as-a-service) palveluna internetissä. Vuoteen 2006 mennessä luotiin Amazon Web Services (AWS) ja julkaistiin Elastic Compute Cloud (EC2) -palvelu. Palvelun avulla asiakkaat voivat vuokrata virtuaalikoneita heidän sovellusten infrastruktuuriksi. Samana vuonna Google julkaisi Google Docsin, jota käytetään asiakirjojen luomiseen, muokkaamiseen ja jakamiseen pilvessä.[ https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0178-3]

Vaikka pilvipalveluiden kasvu alussa on ollut hidasta, viimeisten 10 vuoden aikana pilvipalvelut ovat laajentuneet merkittävästi. Vuoteen 2010 mennessä Amazon, Google, Microsoft ja OpenStack olivat kaikki käynnistäneet pilviosastot. Tämä auttoi saamaan pilvipalvelut yleisesti saataville. Sen jälkeen pilvipalvelut ovat vallanneet suuren osan teknologiateollisuudesta ja pilvisiirrot tai -migraatiot ovat yleistyneet. [https://www.dataversity.net/how-the-cloud-has-evolved-over-the-past-10-years/]

Pilvilaskenta-alustat voidaan jakaa kolmeen ryhmään: julkiseen, yksityiseen ja hybridiin. Julkiset alustat ovat yleisin pilvipalveluiden alusta. Pilviresurssit (kuten palvelimet ja tallennustila) ovat kolmannen osapuolen pilvipalveluntarjoajan omistamia ja ylläpitämiä, ja ne toimitetaan Internetin kautta. Julkisessa pilvessä kaikki laitteistot, ohjelmistot ja muut tukevat infrastruktuurit ovat pilvipalveluntarjoajan omistuksessa ja hallinnassa. Microsoft Azure ja AWS ovat esimerkkejä julkisesta pilvestä. Julkisessa pilvessä jaat samat laitteistot, tallennustilat ja verkkolaitteet muiden organisaatioiden tai käyttäjien kanssa ja käytät palveluita ja hallitset tiliäsi verkkoselaimella. Yksityinen pilvi koostuu pilvipalveluista, joita käyttää yksinomaan yksi yritys tai organisaatio. Yksityinen pilvi voi sijaita fyysisesti omassa palvelinkeskuksessa tai sitä voi isännöidä kolmannen osapuolen palveluntarjoaja. Yksityisessä pilvessä palvelut ja infrastruktuuri ylläpidetään aina yksityisessä verkossa ja laitteistot ja ohjelmistot on omistettu vain yhdelle organisaatiolle. Yksityinen pilvi voi helpottaa organisaation mukauttaa resurssejaan vastaamaan tiettyjä IT-vaatimuksia. Hybridipilvi on alusta, joka yhdistää paikallisen infrastruktuurin tai yksityisen pilvialustan julkiseen pilvialustaan. Hybridialustat mahdollistavat tietojen ja sovellusten liikkumisen kahden ympäristön välillä. Monet organisaatiot valitsevat hybridipilvilähestymistavan liiketoiminnan tarpeiden vuoksi, kuten säännösten ja tietojen riippumattomuusvaatimusten täyttämisen, paikan päällä tapahtuvan teknologiainvestoinnin täyden hyödyn hyödyntämisen tai alhaisen viiveen ongelmien ratkaisemisen vuoksi.

# Teoria

## Tiedostojen siirto

Tässä luvussa käsitellään protokollia, joiden avulla tiedostoja voidaan siirtää pilvialustoille. Ensimmäisessä aliluvussa käsitellään selainten mahdollistamia lataustapoja ja sen jälkeen siirrytään protokolliin, joita erilliset asiakasohjelmat tarvitsevat.

### HTTP ja WebSocket

Selainta voidaan käyttää tiedostojen siirtämiseen TCP-protokollan päällä toimivien protokollien avulla. Tiedostot ovat mahdollista siirtää selaimesta HTTP-protokollan avulla tai WebSocket-protokollalla. HTTP on yksisuuntainen protokolla, jossa asiakas tekee pyynnön palvelimelle ja palvelin vastaa pyyntöön. WebSocket luo kaksisuuntaisen yhteyden asiakkaan ja palvelimen välillen, ja mahdollistaa sen, että palvelin lähettää dataa asiakkaalle ilman erillistä asiakkaan pyyntöä.

Kuvassa 1 on esitelty WebSocketin arkkitehtuuri korkealla tasolla. Aluksi kättelyvaiheeseen käytetään HTTP – protokollaa jonka jälkeen siirrytään WebSocket -protokollaan. Websocket-protokolla muistuttaa TCP-protokollaa, mutta protokollat eroavat kuitenkin toisistaan ja WebSocketilla ei voi muodostaa yhteyttä TCP-serveriin. [1]



Kuva 1 webSocket-protokolla

Asiakas aloittaa kättelyprotokollan lähettämällä HTTP GET pyyynnön palvelimelle. Pyynnössä pitää olla mukana tietyt HTTP-otsikot, jotta palvelin hyväksyy pyynnön. Upgrade-otsikko pitää olla ”websocket” ja Connection-otsikko pitää olla ”Upgrade”. Sec-websocet-version pitää olla 13, joka on nykyinen versio WebSocket standardista. Web-Socket-Protocol otsikko voidaan lisätä ja se kertoo mitä protokollia asiakas tukee. Origin-otsikko lisätään tietoturvan takia ja sen avulla palvelin voi estää luvattomat cross-origin pyynnöt ja se voi hylätä kättely-yritykset listattuihin domaineihin. Pyynnössä on myös mukana Sec-WebSocket-Key otsikko, jonka arvo on satunnainen 16-tavuinen merkkijono, joka on base-64 koodattu. Tämän merkkijonon generointi pitäisi olla kryptografisesti satunnainen ja sen arvoja ei pitäisi pystyä arvaamaan. Jos palvelin hyväksyy kättelyn, niin pyynnön HTTP-vastauskoodi on 101 ja vastauksessa on seuraavat otsikot: Upgrade: ”websocket” ja Connection: ”Upgrade”. Sec-WebSocket-protocol-otsikon arvo on palvelimen valitsema protokolla. Sec-WebSocket-Accept otsikon arvo riippuu pyynnön Sec-WebSocket-Key otsikosta. Palvelin lisää Sec-WebSocket-Key otsikkoon joka kerta vakio GUIDIn 258EAFA5-E914-47DA-95CA-C5AB0DC85B11 ja laskee niiden summasta SHA1-tiivisteen. Tämän tiivisteen palvelin palauttaa Sec-WebSocket-Accept otsikossa. Jos Sec-WebSocket-Accept otsikossa palautuu väärä arvo tai HTTP-pyynnön paluukoodi ei ole 101, asiakas ei avaa yhteyttä.

Kun kättely on onnistuneesti tehty, siirrytään WebSocket-prokollaan. Protokollassa data liikkuu kehyksissä, joiden mallikuva on kuvassa 2. Kehyksessa olevilla numeroilla on seuraavat merkitykset alkaen ensimmäisestä bitistä:

* FIN, 1 bitti: merkkaa onko kehys lähetettävän datan viimeien vai ei. 1 jos on, 0 jos ei ole
* RSV1, RSV2, RSV3 1 bitti kukin. Kaikki ovat nollia jos ei ole lisäosaa kommunikoinnin alussa sovittu.
* Opicode, 4 bittiä. Koodi joka merkkaa kehyksen tyypin
* MASK, 1 bitti. Merkkaa onko siirrettävä data naamioitu vai ei. Jos asiakas lähettää kehyksen tämä bitti täytyy olla 1
* Siirrettävän datan pituus, 7 bittiä. Jos tämä luku on 0-125 se merkkaa siirrettävän datan pituuden. Jos se on puolestaan 126 se tarkoittaa että seuraavat 16 bittiä muostaa luvun, joka on siirrettävän datan pituus. Jos luku on 127 seuraavat 64 bittiä muodostaa luvun, joka on siirrettävän datan pituus. Tämän luvun merkitsevin bitti täytyy olla kuitenkin nolla.
* naamiointiavain, 32 bittiä jos Mask bitti on 1. Tällä avaimella asiakkaan on naamioitava data
* Siirrettävä data, jonka pituus on ilmoitettu siirrettävän datan pituus-arvolla. Se pitää sisällään lisäosa-datan sekä sovellus-datan. Lisäosa-datan pituus on 0 jos lisäosaa ei ole käytössä, muuten lisäosa määrittää tämän datakentän pituuden. Sovellus-datan pituus on siirrettävän datan pituus miinus lisäosa-datan pituus.



Kuva 2: WebSocket kehys

Kun asiakas lähettää kehyksen palvelimelle, MASK bitillä täytyy olla arvo 1, ja 32-bittinen naamiointiavain määriteltynä. Avaimen pitää olla jokaiselle kehykselle eri ja sen generoivan algoritmin kryptografisesti vahva, jotta tulevia arvoa ei voida arvata vanhojen arvojen pohjalta. Datan naamioiminen tapahtuu seuraavalla tavalla 8 bitin lohkoissa: Merkataan lohkon indexiä :llä ja avaimen indeksiä :llä. Indeksit kähtevät nollasta liikkeelle seuraavalla notaatioilla tarkoitetaan 8 bitin lohkoa, joka alkaa indeksin j osoittamasta paikasta. Indeksi kasvaa aina yhdellä siirryttäessä seuraavaan lohkoon ja lohkon aloitusbitti saadaan kertomalla inedeksi kahdeksalla. Avaimen -ideksi saadaan seuraavalla kaavalla

Modulo joudutaan ottamaan kun avaimen pituus on aina 32 bittiä ja datan pituus voi olla pidempi. Siirrettävä naamioitu data saadaan ottamalla XOR operaatio avaimesta ja naamioimattomasta datasta

Datan naamioimisen tarkoitus on tehdä WebSocket-liikenteestä HTTP-liikenteestä poikkeavaa ja arvaamatonta. Muuten verkkoinfrastruktuurilaitteet, joita ei ole päivitetty ymmärtämään WebSocket-liikennettä voivat luulla sitä virheellisesti normaaliksi HTTP-liikenteeksi ja avata tietoturva-aukkoja. Naamioimisella pystytään suojaamaan verkkoinfrastruktuurilaitteet, ja vaikka hyökkääjä pystyisi hallitsemaan asiakaspuolen sekä palvelimen koodia, se ei pysty hallitsemaan liikutettavan kehyksen arvoja, koska ne on naamioimisen ansiosta satunnaisia.

4-bittisellä Opcode-koodilla pystytään merkkaamaan kehyksen tyyppi. Kehys voi olla datan siirtoon tarkoitettu kehys tai hallintakehys. Jos kehys on hallintakehys, sillä on sovittu hallintakoodi Opicode lohkossa ja mahdollisesti siirrettävää dataa. Mahdollisia komentoja on ”close”, ”ping” ja ”pong”. Close-komento aloittaa yhteyden sulkemisprosessin ja se voi sisältää datakentässä viestin, miksi yhteys halutaan sulkea. Kun Close-komento on vastaanotettu, siihen vastataan lähettämällä kehys, jonka komento on myös Close. Close-vastausta voi viivästyttää ja jatkaa datan lähettämistä, mutta silloin ei ole taetta että Close komennon lähettänyt osapuoli pitää yhteyden auki. Ping-komennon tarkoitus on tarkistaa että yhteyden toinen pää ottaa vielä vastaan liikennettä. Kun Ping-komento on vastaanotettu, siihen vastataan lähettämällä Pong-komento. Jos kehys on datan siirtokehys Opcode-koodilla hallitaan siirrettävän datan formaattia. Mahdollisia vaihtoehtoja ovat binääri ja teksti. Tekstin käyttämä merkistö on UTF-8. Opcoden avulla voidaan pilkkoa lähetettävä data useampaan kehykseen, jolloin kaikkien keysten datan formaatti pitää olla sama kuin ensimmäinen kehyksen. Ensimmäinen kehys määrittää Opcodella siirrettävän datan tyypin ja seuraavien kehysten Opcode määrittää että se on jatkoa edelliselle kehykselle. Datan viimeisen kehyksen FIN-bitin arvo on 1, joka kertoo että kyseessä on datan viimeinen kehys. Opicode-koodissa on vapaita komentoja lisäosia varten, joita ei ole määritelty WebSocket-protokollaan. Asiakas voi pyytää lisäosan käyttöä kättelyvaiheessa lisäämällä Sec-WebSocket-Extensions otsikon pyyntöön. Jos serveri hyväksyy lisäosia käyttöön, se lisää saman Sec-WebSocket-Extensions otsikon vastaukseen, johon se listaa hyväksytyt lisäosat.

### FTP ja FTPS

FTP eli File Transfer Protocol on tiedostojen siirtoon käytetty protokolla. Protokolla toimii TCP/IP-yhteyden päällä ja se koostuu asiakkaasta ja palvelimesta. Kuvassa 2 on esitetty FTP protokollan arkkitehtuuri korkealla tasolla. Sekä asiakas että palvelin tarvitsevat ohjelmat, jotka toteuttavat protokollan. Asiakkaan ohjelma voi olla graafinen käyttöliittymä tai se voi toimia komentoriviltä. [2]



Kuva 3 FTP-protokolla

Protokolla käyttää kahta rinnakkaista TCP yhteyttä: hallintayhteyttä ja datayhteyttä. Hallintayhteydellä lähetetään kaikki komennot liittyen käyttäjien hallintaan ja tiedonsiirtoon, ja se on totetutettu Telnet-protokollalla. Telnet on vanha sanomamuotoinen protokolla, joka toimii TCP-prokollan päällä. Telnetillä pystyy muodostamaan yhteyden etätietokoneelle ja sitä on käytetty aikoinaan mm. sähköpostien lähetykseen. Telnet ei salaa sanomiaan, minkä vuoksi protokolla on tietoturvaltaan heikko. Sen käytöstä on luovuttu moderneissa järjestelmissä ja se on korvattu turvallisimmilla protokollilla, kuten SSH. Hallintayhteydellä määritellään datayhteyden ominaisuudet ja sillä avataan haluttu datayhteys. [2]

Datayhteydellä siirretään tiedostot. Yhdellä aktiivisella yhteydellä voidaan siirtää vain yhtä tiedostoa samanaikaisesti. Datayhteys siirtää datan aina 8-bittisinä tavuina joita kutsutaan siirtotavun kooksi. FTP tukee muutamaa eri tiedonsiirtotyyppiä, joina se pystyy esittämään siirrettevän datan. Oletustyyppi on ASCII, jota kaikkien FTP-toteusten pitää tukea. ASCII on tekstin merkistö, jossa jokainen merkki esitetään 7 bittisinä numerona. Tiedosto lähetetään 8-bittisenä NVT ASCII:na jossa eniten merkitsevä bitti on nolla ja sen perässä 7-bittinen ASCII koodi. Vaihtoehtoinen tyyppi lähettää tekstitiedostoja on EBCDIC. Se on 8-bittinen merkistö, joka on pääasiassa käytössä IBM:n suurtietokoneissa. ASCII ja EBCDIC ottavat myös toisen vapaaehtoisen parametrin joka määrittää tekstin pystysuuntaista asettelua ja sitä käytetään tekstin tulostuksen muotoiluun. Oletusarvo Non-print ei ota kantaa muotoiluun. Telnet format control parametriksi annettuna olettaa että tekstistä löytyy vertikaalisia tekstin hallinta komentoja kuten LF ja NL. Carriage control olettaa että tekstistä löytyy ASA (FORTRAN) vertikaali hallinta merkkejä ja hallitsee niiden avulla tekstin muotoilua. Image-tyyppi on binääritiedostojen lähettämiseen tarkoitettu tyyppi. Tiedosto luetaan biteiksi ja bitit lätetään 8-bittisinä tavuina. Lähetyksen loppuun saatetaan tarvita täytetttä jos tiedoston bitit eivät ole kahdeksalla jaollisia. Täyte on pelkästään nollia, ja asiakkaan ja palvelimen täytyy sopia kuinka täyte tunnistetaan oikeasta datasta. Local-tyyppi on image-tyypin laajennos ja sen avulla voidaan muuttaa lähetettävän datan loogisen tavun kokoa annettavalla parametrillä. [2]

FTP tukee kolmea eri tiedostonvälitysrakennetta. Tiedostorakenne, tallennerakenne ja sivurakenne: Tiedostorakenteessa tiedostolla ei oleteta olevan mitään sisäistä rakennetta, vaan tiedosto lähetetään binäärimuodossa. Tallennerakenteessa tiedoston oletetaan koostuvan toistuvista pienemmistä osista. Sivurakenteessa tiedoston oletetaan koostuvan sivuista, jotka ovat indeksoitu. Tätä rakennetta käytetään epäjatkuvien tiedostotyyppien kanssa jotka voivat sisältää useampia sisäisiä tiedostoja jotka tunnistetaan tiedostossa olevalla otsikkodatalla. [2]

Datan siirtämisen aloittaminen edellyttää datayhteyden muodostamista sekä siirtoparametrien sopimisesta. Aluksi asiakkaan pitää avata datayhteyden portti kuuntelemista varten ja lähettää pyyntösanoma hallintayhteydellä. Tämä sanoma pitää sisällään tiedon kumpaan suuntaa dataa ollaan siirtämässä. Palvelin vastaa pyyntöön ottamalla yhteyden asiakkaan datayhteyden porttiin. Seuraavaksi tiedoston siirtotapa pitää sopia. FTP tukee kolmea tapaa: stream, block ja Compressed. Stream lähettää tiedostot jatkuvana virtana. Jos tiedoston tiedostonvälitysrakenne on tiedostorakenne, niin tiedoston päättyminen merkataan sulkemalla datayhteys. Tallennerakenteessa käytetään kaksitavuista hallintakoodia merkkaamaan osien päättymistä tai tiedoston päättymistä. Block-siirrossa data siirretään lohkoina. Kullakin lohkolla on 3-tavuinen otsikko, jossa on tieto lohkon pituudesta sekä lohkon tyypistä. Lohkon tyypillä pystytään merkkaamaan lohkoon tieto siitä, että onko lohkotiedoston viimeinen, tallenteen viimeinen vai uudelleenkäynnistyksen merkki. Compressed-siirto käyttää yksinkertaista pakkausalgoritmia nimeltään run-length encoding joka lähettää datan lohkoissa kuten block-siirrossa. Pakkaamalla data siirrossa saadaan pienennettyä siirrettävän datan määrää, mutta yleensä pakkaus tehdään tehokkaimmilla algoritmeilla jo tiedostoa luodessa tai verkkotasolla modeemeiden toimesta. FTP-protokolla ei sisällä tapaa tarkistaa siirrettävän datan eheyttä, vaan se luottaa TCP-protokollaan. FTP sisältää kuitenkin protokollan tiedoston latauksen jatkamiseen, jos lataus on keskeytynyt. Latauksen jatkamista voi käyttää vain jos siirtotapa on block tai compressed. Latauksen jatkaminen merkataan lohkon otsikkoon sille varatulla koodilla paikassa, josta tiedoston lataus jatkuu. [2]

FTP-protokolla ei salaa kumpaakaan käyttämäänsä yhteyttä joten sitä pidetään nykyään tietoturvattomana. FTPS (FTP Secure) on FTP-protokollan tietoturvallisempi lisäosa, joka käyttää SSL- tai TLS-suojausprotokollaa. Suojauksen päälle kytkeminen voidaan toteuttaa kahdella eri tavalla: implisittisesti ja eksplisiittisesti. Implisiittinen tapa vaatii suojausta jo heti yhteyden luomisvaiheessa. Tämä rikkoo pelkästään FTP-protokollalla toimivien asiakasohjelmien ja palvelimien toimivuuden implisiittisesti suojausta vaativan vastinparinsa kanssa. Eksplitisiittisessä suojauksessa asiakkaan täytyy erikseen pyytää suojattua yhteyttä. Näin palvelin mahdollistaa kommunikoinnin sekä FTP- että FTPS-asiakkaiden kanssa. Eksplisitiittisesti määriteltäessä voidaan myös säätää mitkä yhteydet suojataan, kun implisiittisesti määriteltynä molemmat kanavat ovat suojattuna. Esimerkiksi jos siirrettävä data on jo valmiiksi salattu, pelkästään hallintayhteys voidaan salata. [2]

### SFTP

SFTP-protokolla on tiedostonsiirto- ja tiedostonhallintaprokolla, joka koostuu palvelimesta ja asiakasohjelmasta joka ottaa yhteyden palvelimeen. SFTP-prokolla toimii suojatun turvallisen kanavan avulla ja se suunniteltiin SSH 2-protokollan laajennokseksi. SFTP prokolla ei itsessään suojaa liikennettä, vaan se luottaa alla olevan kanavan suojaukseen. SFTP-yhteyden avulla voidaan siirtää tiedostoja ja tehdä tiedostonhallintaa. Vaikka SFTP-protokolla suunniteltiin SSH 2-prokollan laajennukseksi, sitä voidaan käyttää myös muiden protokollien avulla esimerkiksi SSH-1 protokollan avulla. SSH 1-prokollan käyttö aiheuttaa rajoitteita asiakasohjelmalle, koska sen pitää tietää SFTP-palvelimen binääritiedostojen sijainti palvelimella. SFTP-protokollasta on julkaistu useita versioita, joista käytetyin versio nykyään on versio 3 joka julkastiin vuonna 2001. Vuoden 2001 jälkeen protokollasta on julkaistu versiot 4, 5 ja 6, mutta ne eivät ole laajalti tuettuja ja useat SFTP-palvelimien kehittäjät ovat ilmoittaneet, että heillä ei ole aikomustakaan tukea uusimpia versioita tulevaisuudessa.

SFPT-protokolla lähettää paketit verkon yli muodossa, jossa on alussa tavun kokoinen tila komennolle ja sen jälkeen tulee siirrettävä data. Kuvassa 4 on esitetty SFTP-protokollan arkkitehtuuri, ja siinä näkyy pakettien komentojen nimet. Asikas lähettää aluksi SSH\_FXP\_INIT-komennon jonka datana on haluttu versio protokollasta. Palvelin vastaa SSH\_FXP\_VERSION-komennolla, joka pitää sisällään palvelimen tukemat versiot. Palvelimen ja asiakasohjelman pitää tukea samaa versiota protokollasta, jotta yhteys pystytään muodostamaa. Tiedostojen käsitteleminen tapahtuu niille tarkoitetuilla komennoilla. Esimerkiksi SSH\_FXP\_OPEN komento avaa tiedoston tai luo uuden tyhjän tiedoston. Palvelin palauttaa SSH\_FXP\_OPEN kyselyyn SSH\_FXP\_HANDLE komennon jos SSH\_FXP\_OPEN kysely oli onnistunut. SSH\_FXP\_HANDLE palauttaa datana ID:n jonka avulla tiedosto voidaan siirtää asiakasohjelmalle SSH\_FXF\_READ lukukomennon avulla. Esimerkkejä muista komennoista : SSH\_FXF\_WRITE mahdollistaa tiedostojen kirjoittamisen palvelimelle, SSH\_FXP\_REMOVE mahdollistaa tiedostojen poistamisen palvelimelta. Diagram

Description automatically generated

Kuva 4 SFTP-protokolla

## Tiedostojen talletus

Tässä luvussa käsitellään eri tallennusmuotoja, joita pilvialustat yleisesti tarjoavat.

### SQL-tietokannat

SQL-tietokannat ovat relaatiotietokantoja, joissa tieto on tallennettuna tauluina, jotka koostuvat riveistä ja sarakkeista. Taulut perustuvat relattiomalliin, jonka määrittää predikaattilogikka. Taulu esittää yhden asian tiedot, sarakkeet kuvaavat kokonaisuuden arvoja ja rivit asian ilmentymiä. Tauluissa jokaisella rivillä on uniikki ID, joka pystyy muodostumaan joko yhdestä tai useammasta ssarakkeen arvosta. Taulujen välille pystytään luomaan yhteyksia toisiinsa taulujen avainten avulla.

Tietokannan käytön mahdollistavat tietokannan ohjausjärjestelmät. Ne mahdllistavtavat käyttäjän pääsyn tietokantaan sekä sen päivittämisen ja muokkauksen. Tietokannan ohjausjärjestelmiä on maksullia ja avoimen lähdekoodin järjestelmiä. Suosituimmilla avoimen lähdekoodin ohjausjärjestilmillä on hyvin samllainen pääarkkitehtuuri, mutta toteutuksissa on pieniä eroavaisuuksia. Kuva 5 esittää Postgres tietokannan arkkitehtuuria. Postgres on yksi suosituimmista ilmaisen lähdekoodin tietokannosta ja se on julkaistu vuonna 1995. Postgres toimii asikas-palvelin -mallilla, jossa asikasohjelma ottaa yhteyden ohjausjärjestelmään ja se luo jokaisella asikasyhteydelle oman palvelinprosessin.Kaikilla palveluprosesseilla on pääsy jaettuun muistiin joka on RAM -muistissa. Tietokantakyselyt tehhdään jaettuun muistiin, joka on kopio levyllä olevasta tiedosta. Jaettumuisti on huomattavasti nopeampaa kuin levyltä luku. Lukuoperaatiot kohdisttuvat pelkästään jaettuun muistiin, mutta kirjoitus operaatiot tehdään WAL(Write ahead log) logiin. Ensiksi muutokset tehdään jaettuun muistiin, jonka jälkeen WAL-kirjoittaja – prosessi kirjoittaa muutokset levylle säännöllisenä taustaprosessina. Checkpointer ja writer prosessit päivittävät muutoksetjaetusta muistista levylle sen jälkeen kun muutokset ovat tehty WAL- tiedostoon. Levylle kirjoitus tehdään taustaprosessina pienissä paloissa koska I/O -operaatio on raskas. Jaetusta muistissa pidetään tallessa poistettuja rivejä sekä muuttuneiden rivien vanhoja versioita, koska toiset palveluprosessit saattavat tarvita vanhoja arvoja. Autovacuum – prosessi poistaa vanhentuneet rivit jaetusta muistista. [https://www.postgresql.org/docs/current/routine-vacuuming.html, https://en.wikibooks.org/wiki/PostgreSQL/Architecture] Diagram

Description automatically generated

Kuva 5 Postgresin arkkitehtuuri

### NOSQL

NoSQL tarjoaa mahdollisuuden tallentaa datan tietokantaa muuna rakenteena kuin relaatiomallin tauluina. Ensimmäiset NoSQl tietokannat kehitettiin jo 1960 -luvulla, mutta nimitys NoSQL syntyi 2000 -luvun alussa. NoSQL kannoille ei ole yhtä määritelmää vaan ne jakautuvat useampaan kategoriaan. Yleisinpiin kategorioihin kuuluu mm. erillaiset avain-kuorma- tietokannat, jotka käyttävät hakemisto-tietotyyppiä tiedon tallettamiseen. Data talletetaan tietokantaan uniikin avaimen avulla, johon on linkitetty sitä vastaava kuorma. Toinen yleinen NoSQL tietokanta on dokumentti – tietokanta, joka tallettaa tiedon jossakin standardoitusa tiedostoformaatissa esimerkiksi JSON- tai XML – formaatissa. Dokumentit ovat talletettuna tietokantaan uniikin ID avulla joka tekee dokumentti – tietokannoista avain-kuorma -tietokannan osajoukon.

### Object Storage

Object Storage tarjoaa tallennusmahdollisuuden rakenteettomalle datalle. Tiedostojen tyypillä ja koolla ei ole mitään merkitystä vaan kaikki tiedostot tallennetaan yhteen ämpäriin. Ämpärissä tiedostoilla ei ole minkään sortin kansiorakennetta, vaan jokaisella tiedostolla on ämpärissä uniikki id. Ämpäriin pääsee käsiksi HTTP REST-rajapinnan kautta, joka mahdollistaa tiedostojen hallinnan. Jokainen Object Storagen toteuttava taho voi luoda oman REST-rajapinnan, mutta käytännössä kaikki pilvialusta toteuttavat Amazon AWS S3 ämpärin rajapinnan. S3 oli ensimmäinen Object Storage pilvialustoilla ja sen rajapinta oli hyvin dokumentoitu, jonka seurauksena siitä tuli erittäin suosittu ja muutkin pilvialustat alkoivat tarjoamaan samaa rajapintaa. Object Storage tarjoaa erittäin hyvän skaalautuvuuden, koska ämpärin kokoa voidaan kasvattaa erittäin suureksi, jopa exatavujen suuruiseksi. Object Storage on myös erittäin kustannustehokas ratkaisu, kun vain käytetystä talletustilasta laskutetaan eikä yhtään ylimääräisestä varalla olevasta talletustilasta kuten normaaleissa levytalletusratkaisuissa yleensä joutuu maksamaan.

Nykyään jokaisella suurella pilvialustalla on oma versio Object Storagesta ja niiden sisäiset toteutukset voivat vaihdella. Seuraavaksi on esitelty IBM:n toteutus asialle ja Objectect Storagen arkkitehtuuri on esitetty kuvassa 5. Manager tarjoaa toiminnot admin-käyttäjille ja sen avulla voidaan valvoa ja muokata järjestelmää. Accesser tarjoaa REST-rajapinnan käyttäjille, sekä hoitaa datan salauksen kirjoittaessa ja salauksen purun lukuvaiheessa. Slicestor on vastuussa tiedostosiivujen tallentamisesta, jotka se saa Accesserilta. Slicestoreiden ryhmää kutsutaan Device Setiksi ja Device Settien ryhmää puolestaan Storage pooliksi. Storage poolit voivat koostua Device Seteista, jotka sijaitsevat eri palvelinkeskuksissa. Ämpärit kuvataan sisäisessä arkkitehtuurissa Voulteina ja ne sijaitsevat Storage poolissa. Yksi Storage pool voi pitää sisällään useita Voulteja ja jokainen Voult pitää sisällään Slicestoreita jokaisesta Device Setistä

. 

Kuva 6 IBM Object Storage

Datan tallentamiseen käytetään Information Dispersal Algoritmia (IDA). Algoritmin perusidea on seuraava: Tallennettava datasta luodaan N kappaletta paloja. Tätä lukua kutsutaan algoritmin leveydeksi. Kuitenkaan kaikkia näitä paloja ei tarvita, jotta tiedosto pystytään lukemaan tai kirjoittamaan kokonaisuudessan, vaan pienempi määrä paloja riittää. Read Treshold kertoo määrän kuinka monta palasta tarvitaan alkuperäisen datan lukemiseksi ja Write Threshold kuinka monta palasta tarvitsee olla kirjoitettuna, jotta kirjoitusoperaatio merkataan suoritetuksi. Write Threshold on aina suurempi mitä Read Treshold.

Kuvassa 6 on esitetty tiedoston kirjoitusprosessi. Prosessi lähtee likkeelle Accesser Nodeista, jotka pilkkovat yli 4 MiB kokoiset tiedostot neljäksi erilliseksi siivuksi. Siivut viedään yksitellen loppuprosessin läpi tai pieni tiedosto viedään kerralla kokonaisena. Seuraavaksi dataan lisätään eheyden tarkistusta varten tarkistusarvo tiivistefunktiolla ja data voidaan salata lohkosaluksen avulla. Data viipaloidaan ensiksi Read Treshold arvon mukaisesti viipaleista ja näistä viipaleista luodaan algoritmin leveyden verran paloja joista tarvitsee kirjoittaa algoritmin Write Threshold- määrän mukaisesti paloja, jotta kirjoitusoperaatio merkataan suoritetuksi. Nämä palat lähetetään Slicestor Nodeille tallennettavaksi. Kun kirjoitusoperaatio on merkattu onnistuneesti suoritetuksi, puuttuvat palaset kirjoitetaan asynkroonisesti taustaprosessina. Lukuprosessi on päinvastainen suhteessa kirjoitusprosessiin. Slicestor Nodeilta luetaan Read Treshold- määrän mukainen määrä siivuja ja näistä siivuista palautetaan alkuperäinen tiedosto.



Kuva 7 Tiedoston kirjoitusprosessi

### Block storage

Block storage tallentaa datan vakiokokoisina paloina, jotka se tallentaa yksilöllisen tunnisteen avulla. Tavallinen tietokoneen kiintolevy toimii käyttöjärjestelmässä Block storagen kanssa identtisesti tiedostoja tallentaessa ja tiedostot ovat pilkottu palasiksi kiintolevyllä. Block storage toteutetaan pilvialustoissa tallennusalueverkkojen (SAN) avulla, jotka ovat yleisimpiä varastointiverkkoarkkitehtuureja. SAN yhdistää useita tallennuslaitteita yhdeksi verkoksi, johon tallennustilaa käyttävät laitteet myös liittyvät.

SAN mahdollistaa verkon luomisen muutamalla eri protokollalla, mutta kaikki prokollat käyttävät SCSI-protokollaa. SCSI on vanha, vuonna 1986 luotu protokolla, joka määrittää tiedonsiirtoväylän sekä mitenkä siihen kytketyt laitteet kommunikoivat keskenään. Useat tietokonearkkitehtuurit sisältävät erillisen väylän I/O operaatioille, joiden avulla CPU juttelee I/O laitteiden kanssa. Tämä väylä toteuttaa SCSI-protokollan. SCSI-väylällä voidaan yhdistää useita tallennuslaitteita ja palvelimia, mutta sen skaalautuvuus on erittäin heikko. Luotuun verkkoon voidaan lisätä vain rajallinen määrä laitteita, sekä kaapelin pituudella on maksimi pituus. Versiosta riippuen verkkoon voidaan lisätä 8-16 laitetta sekä laitteet voivat olla toisistaan vain noin 20 metrin päässä toisistaan. Koska SCSI-väylät eivät ole järkeviä tallennusalueverkkojen luomista varten ne on korvattu järkevimmillä väylillä.

Valokuituverkot käyttävät valokuituja tiedonsiirtoon ja Fibre Channel Protocol (FCP) on verkon protokolla. Valokuituverkot ottavat vastaan SCSI-komentoja ja muuttavat ne FCP-protokollan komennoiksi valokuiduissa tapahtuvan siirron ajaksi ja muuttavat ne takasin SCSI-komennoiksi kun siirto on tehty. Tätä varten tarvitsee asentaa oma laiteajurinsa käyttöjärjestelmään. Tallennustilaa käyttävät laitteet ja tallennuslaitteet kytketään samaan valokuituverkkoon, jolloin tiedostonsiirrot eivät rasita normaalia intenetyhteyttä. Tämä prokolla on eniten käytetty ja nopein, koska valokuituverkko on erittäin nopeaa tiedostojen siirtoon. Haittapuolena on verkon erittäin korkea hinta ja verkon ylläpitämiseen tarvittava erikoisosaaminen. Toisista yleisin protokolla on iSCSI, joka käyttää internetyhteyttä valokuituverkon sijasta, ja on huomattavasti halvempi kuin valokuituverkot, mutta hitaampi. iSCSI siirtää SCSI-komennnot TCP-protokollan avulla verkossa. iSCSI on myös tehottomampi tapa siirtää dataa, koska TCP-pyynnöissä on paljon enemmän turhaa dataa mukana suhteuttuna SCSI-komentoihin verrattuna FCP-protokollaan. Muita TCP:tä käyttäviä SAN-protokollia ovat: iFCP ja FICP, mutta ne ovat huomattavasti vähemmin käytetty mitä iSCSI.

Pilvipalveluiden tarjoamat Block storaget ovat lähes poikkeuksessa toteutettu iSCSI protokollan avulla ja ne voi sitoa virtuaalikoneisiin tavallisena kansioina. Block storageihin tallennettu data ei ole sidottu virtuaalikoneen elinkaareen ja data säilyy vaikka virtuaalikoneen poistaisi mihin Block storage oli sidottu.

### File storage

File storage tallentaa datan hierarkisessa rakenteessa käyttäen tiedostoja ja kansioita. File storage mahdollistaa levyn käytön usealle yhtäaikaiselle käyttäjälle ja sen toteutus perustuu samoihin teknologioihin kuin tavallisen verkkolevyn. Yleisin teknologia on NAS, joka on esitetty kuvassa 7. NAS käyttää sisäisesti samoja teknologioita kuin SAN, mutta se lisää oman protokollakerroksen niiden päälle hallitsemaan tiedostoja. NAS-palvelin avaa IP-osoitteen, mihin asiakkaan voivat ottaa yhteyden LAN-verkosta. Tällöin tiedostot liikkuvat samassa verkossa, missä muukin internetliikenne tapahtuu ja tiedostojen siirto voi rasittaa muuta internetliikennettä. NAS edellyttää asiakkailta erillistä asikasohjelmaa, joka on käyttöjärjestelmästä riippuvainen. Yleisin asikasjärjestelmän protokolla on NFS.



Kuva 8 NAS-arkkitehtuuri

## Pilvialustat

Tässä kappaleessa käsitellään projektissä läytettävät julkisen pilven palvelut.

### Microsoft Azure

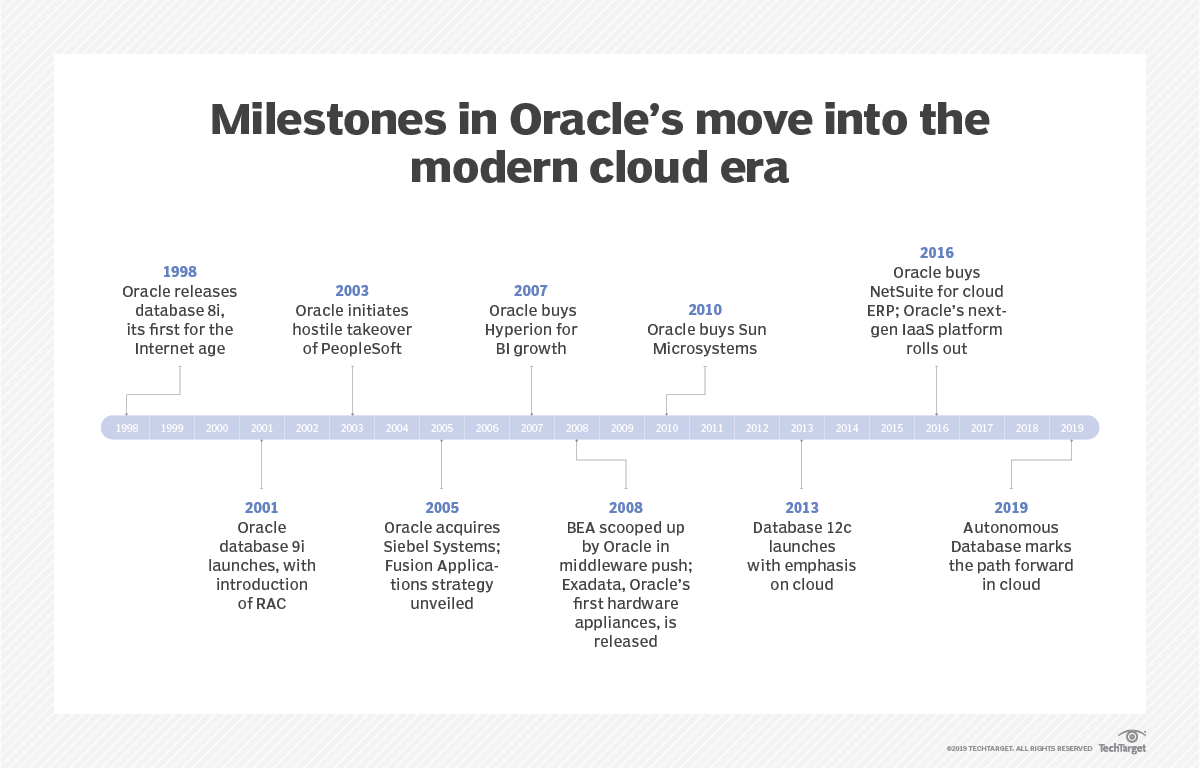
Microsoft Azure on Microsoftin ylläpitämä pilvipalvelu, joka julkistettiin Microsoftin Professional Developers Conferencessa (PDC) lokakuussa 2008, ja se julkaistiin virallisesti helmikuussa 2010 nimellä Windows Azure vaihtoehdoksi silloin jo tunnetuille Amazon ja Google pilvialustoille. Azuren ensimmäinen versio tarjosi hyvin rajoitetun joukon palveluita, pelkästään pilvipalvelun ASP.NET-verkkosovellusten kehittämiseen ja käyttämiseen, Azure Blob -tietovaraston, Azure SQL -pilvitietokannan ja Azure Service Busin. Vuonna 2014 sen nimi muutettiin nykyiseksi Microsoft Azureksi. Alustan taustatarina alkoi vuonna 2005, kun Microsoft osti Groove Networksin ja Microsoftin kehittäjät alkoivat kehittämään pilvikäyttöjärjestelmää. Azurea koodattiin aluski koodinimella Red Dog ja se julkistettiin vuonna 2008. Se oli tuolloin Windows NT:n laajennus, joka oli suunniteltu toimimaan pilvessä. Se tarjoaa ohjelmistoa palveluna (SaaS), alustan palveluna (PaaS) ja infrastruktuurin palveluna (IaaS) ja tukee monia erilaisia ​​ohjelmointikieliä, työkaluja ja kehyksiä, mukaan lukien sekä Microsoft-kohtaiset että kolmannen osapuolen ohjelmistot ja järjestelmät.[ https://apix-drive.com/en/blog/reviews/microsoft-azure-review]

Azure, kuten muut pilvialustat perustuu virtualisointiteknologiaan. Tietojenkäsittelyssä virtualisointi tarkoittaa virtuaalisen eikä todellisen version luomista jostakin tietokoneen osasta, mukaan lukien virtuaaliset tietokonelaitteistot, tallennuslaitteet ja tietokoneverkkoresurssit.. Suurin osa tietokonelaitteistoista voidaan voidaan virtualisoida ohjelmistossa. Tietokonelaitteisto on yksinkertaisesti joukko ohjeita, jotka on pysyvästi tai puolipysyvästi koodattu piihin. virtualisointikerroksia käytetään yhdistämään ohjelmistoohjeet laitteiston ohjeisiin. virtualisointikerrokset mahdollistavat virtualisoidun laitteiston suorittamisen ohjelmistossa, kuten itse laitteisto. [https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure]

Pohjimmiltaan pilvialustat ovat joukko fyysisiä palvelimia yhdessä tai useammassa datakeskuksessa. Palvelinkeskukset toteuttavat virtualisoituja laitteistoja asiakkaille. Jokaisen palvelinkeskuksen sisällä on kokoelma palvelimia palvelintelineissä. Jokainen palvelinteline sisältää useita palvelinkortteja ja verkkokytkimen. Nämä tarjoavat verkkoyhteyden ja virranjakeluyksikön (PDU), joka tuottaa virtaa. Telineet on joskus ryhmitelty suuremmiksi yksiköiksi, jotka tunnetaan klustereina. Palvelimen telineet tai klusterit suorittaa virtualisoituja laitteisto-ilmentymiä käyttäjille. Jotkut palvelimet käyttävät kuitenkin pilvihallintaohjelmistoa, joka tunnetaan nimellä kuituohjain. kuituohjain on hajautettu sovellus, jolla on monia vastuita. Se allokoi palveluita, tarkkailee palvelimen ja siinä käynnissä olevien palveluiden kuntoa ja korjaa palvelimia, jos ne kaatuvat. Kukin kuituohjain on yhdistetty pilviorkesteriohjelmistoon jossa pyörii servereitä, Rajapintoja sekä sisäisiä tietokantoja joita Azure vaatii toimiakseen. pilviorkesteriohjelmistossa pyörii muunmuissa palveluita jotka vastaavat käyttäjien tekemiin pyyntöihin hallintakäyttöliittymän kautta. Käyttäjän pyynnöt allokoivat Azure-resursseja ja -palveluita. Ensin käyttöliittymä varmistaa, onko käyttäjällä lupa varata pyydetyt resurssit. Jos näin on, käyttöliittymä tarkistaa tietokannan löytääkseen riittävän kapasiteetin omaavan palvelintelineen, joka kehottaa kuituohjainta varaamaan resurssin [https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure]

### Oracle Cloud

Oracle Cloud on Oraclen kehittämä pilvialusta, joka julkaistiin vuonna vuonna 2016. Kuvassa 9 on esitetty Oraclen tarjoamien tietokantojen ja pilvialustojen historiaa. Oracle Cloudin mahdollistaja voidaan pitää Oraclen tekemiä suuria yrityskauppoja. Vuonna 2008 Oracle osti Exadata yrityksen, joka pakkasi palvelimet, verkot ja tallennustilan mukaan lukien Oracle-tietokannan ja muut ohjelmistot valmiiksi määritettyihin telineisiin. Oston myötä Oracle pyrki ottamaan suurempaa valtaa tietovarastointimarkkinoilla. Vuonna 2010 Oracle osti Sun Microsystems yrityksen. Oracle sai kaupan myötä avoimen lähdekoodin MYSQL tietokannan sekä Java -ohjelmointikielen. Sunin laitteistosta tuli alusta Exadatalle ja muille Oraclen palveluille. [https://www.techtarget.com/searchoracle/news/252470532/The-roots-of-Oracles-cloud-evolution-A-20-year-review]



Kuva 9 Oraclen pilvialustojen historia

# Projektin esittely

Tässä kappaleessa esitellään anonyymisti projekti, johon diplomityön käytännön osuus perustuu. Projekti on anomisoitu asiakkaan toiveen mukaan, joten sitä ei tulla mainitsemaan nimellä. Projektin päämääränä on toteuttaa pilvipalvelu, joka mahdollistaa suurien tiedostojen lataamisen ja niiden automaattisen käsittelemisen ja tulosten siirtämisen eri tietovarastoihin. Projektiin valittiin Oracle Cloud -pilvipalvelu asiakkaan toiveesta. projekti on toteutettu mikropalvelu -arkkitehtuurina käyttäen Docker kontteja, joita ajetaan tuotannossa Oracle Cloudin ylläpitämässä Kubernetes -palvelussa .

Tiedostojen lataamiseen valittiin ratkaisuksi selain pohjainen ratkaisu, mutta myös muita vaihtoehtoja tutkittiin. Ohjelman pitää pystyä lataamaan tiedostoja sekä kansioita ja sen pitää pystyä validoimaan, että tiedostot eivät ole korruptoituneet siirron aikana. Latauksen suuruus on luokkaa 1 – 10 Gb, ja lataus pitää pystyä tekemään normaalitehoisella loppukäyttäjän tietokoneella.

Tiedostot ladataan jaettavalle levylle, joka voidaan jakaa useampien konttien kesken. Latauksen jälkeen toinen kontti käsittelee ladattavat tiedostot ja siirtää tulokset eri tietovarastoihin. Käytettäviä tietovastoja on SQL -tietokannat, jaettavat verkkolevyt sekä Object Storage.

Valitut teknologiat valittiin arvioimalla niiden suorituskykyä, hintaa sekä tietoturvallisuutta. Tässä diplomityössä käytännönosuutena on toteutettu pieniä testipenkkejä, jotka mittaavat vaadittuja omanaisuuksia, sekä on vertailtu eri pilvialustojen hintoja.

Seuraavassa luvussa 4 Tulokset käytetään seuraavia ominaisuuksia vertaillessa lataus- ja talletustapoja. Tietoturva, nopeus, käytetty verkkoyhteys latauksen yhteydessä sekä kustannukset.

Tietoturva käsitellään kootusti yhtenä kappaleena jossa vertaillaan käytettyjen protokollien tietoturvallisuutta ja mahdollisia haavoittuvuuksia käytetyissä teknologioissa.

# TULOKSET

Työssä esitellään useampi pienempi testausprojekti, jotka voidaan jakaa karkeasti kahteen osioon: tiedostojen siirtoon sekä tiedostojen talletukseen. Kokonaisjärjestelmän arkkitehtuuri on esitetty kuvassa 8, joka kattaa sekä tiedostojen siirron että talletuksen. Järjestelmän luomiseen on käytetty Azure-pilvipalvelua ja testejä ajetaan Azuren ulkopuolella paikallisella tietokoneella. Azuressa on käytössä virtuaalikone, jossa pyörii .NET:n avulla toteuttu palvelin, Nginx palvelin, joka pyörittää käyttöliittymää, jota käytetään HTTP-latausten testaamiseen. Käyttöliittymä on toteutettu Angular ohjelmointikehyksen avulla ja se on toteutettu Single Page Application (SPA) arkkitehtuurilla. Nginx-palvelinta käytetään SPA-käyttöliittymän jakamiseen internettiin. Virtuaalikoneella on myös asennettu vsftp-palvelin, joka on FTP palvelin, joka mahdollistaa myös FTPS-yhteyden. OpenSSH-palvelinta käytetään SFTP-yhteyden luomiseen. Virtuaalikoneeseen on liitetty Bock ja File storaget ja ne on sidottu osaksi virtuaalikoneen tiedostojärjestelmää ja niihin pääse käsiksi samaan tapaan kuin virtuaalikoneen omaan kovalevyyn. Blob storageen ja Dynamo-tietokantaan pääsee käsiksi julkisen HTTP rajapinnan kautta. Dynamo-tietokanta on Azuren toteuttama NoSQL-tietokanta, jossa data on tallennettuna json-dokumentteina. Virtuaalikoneeen kanssa samassa virtuaalisessa sisäverkossa on myös Postgres SQL-palvelin. Virtaalikone juttelee SQL palvelimen kanssa C# ohjelmoiintikielellä toteutetun asiakasohjelman avulla.

Diagram

Description automatically generated

Kuva 10 Testiohjelman kokonaisarkkitehtuuri

## Tiedostojen siirto

Tiedostojen siirtoon liittyvät projektit ovat jaoteltu useampaan pienempään projektiin eri tiedostonsiirtotapojen perusteella. FTP-, FTPS- ja SFTP-siirrot muodostavat yhden projektin ja selaimella toimivat protokollat muodostavat oman projektin.

FTP, FTPS ja SFTP siirrot ovat toteutettu kahdella eri ohjelmointikielen kirjastolla. Käytetyt ohjelmointikielet ovat Python sekä C#. Python-ohjelmointikielelllä on FTP ja FTPS kirjasto on Pythonin sisäinen kirjasto siihen ei tarvita kolmannen osapuolen kirjastoja. SFTP-yhteys on luotu Paramiko-kirjastolla, joka on kolmannen osapuolen ilmaisen lähdekoodin kirjasto. C#-koodissa on läytetty kolmannen osapuolen Nuget-paketteja yhteyksien luomiseen. FluentFTP-kirjastoa on käytetty FTP- ja FTPS- yhteyksien luomiseen ja SSH.NET-kirjastoa SFTP-yhteyteen. Projektissa on ladattu erikokoisia tiedostoja käyttäen kumpaakin ohjelmointikieltä. Tiedostojen koot ovat olleet kilotavun ja 10 megatavun välissä. Saman kokoinen tiedosto on ladattu viiteen kertaan ja jokaisen latauksen latausaika on kirjattu muistiin. Näistä latausajoista on laskettu keskiarvo kullekin tiedostokoolle ja keskiarvoja on käytetty tulosten käsittelemiseen

Selaimella toimivia protokollia eli HTTP- ja WebSocket-prokollia on testattu selainpohjaisen käyttöliittymän avulla. Kuvassa 9 on näyttökuva käyttöliittymän HTTP-latauksesta

Graphical user interface, text, application

Description automatically generated

Kuva 11 HTTP-latausnäkymä

Käyttöliittymällä pystyy valitsemaan eri asetuksia käytettäviin protokolliin. HTTP-latauksessa tiedostoa ei lueta kokonaan selaimeen muistiin, vaan se luetaan palasina. Jos tiedostoa ei palotelttaisi, koko tiedosto pitäisi lukea selaimen käyttämään RAM-muistiin ja se rajoittaisi merkittävästi lähetettävän tiedoston maksimikokoa. Tiedoston paloittelemisen avulla lähetettävällä tiedostolla ei ole maksimikokoa rajoittavaa tekijää. Käyttöliittymässä on valittavana 5kB-, 50kB- ja 500kB-kokoiset palaset. Kukin palanen läheteään omana POST pyyntönä palvelimellle ja viimeisen palasen jälkeen palvelin kokoaa palaset yhteen ja muodostaa kokonaisen tiedoston palvelimella.Toinen valittava ominaisuus on lähetettävän palasen koodaus. Kaikissa koodauksissa tiedoston palanen luetaan selaimen File API:n avulla ArrayBuffer-tyyppisenä selaimen muistiin. Arraybuffer on datan esitys kahdeksan bittisenä numeroina taulukkotietorakenteessa. Data voidaan siirtää käyttäen eri koodauksia siirrossa. Base64-koodauksessa binäärinen data muunnetaan merkkijonoksi. Base64 koodisto käyttää 64-merkkistä aakkostoa johon binääridata muunnetaan. Number array-valinta lähettää binäärisen datan merkkijonona ilman koodausta. Tämä vaihtoehto kasvattaa huomattavasti siirrettävän tiedoston kokoa, koska jokaisen tavun jokainen numero esitetään UTF-8 merkkinä siirron aikana. Base64 ja Number array siirtävät datan json-formaattina, jonka merkistö on UTF-8. Form-data formaatti siirtää datan binäärimuodossa ja dataa ei tarvitse lähettää merkkijonona.

WebSocket-protokollalla on mahdollista valita siirrettävän datan formaatti. Valittavana on merkkijono sekä binääri. Merkkijono käyttää UTF-merkistöä siirrettävän datan koodauksena ja binääri käyttää binääriformaattia tiedonsiirtoon.

Käyttöliittymää ohjataan testeissä Selenium-kirjastolla, jolla pystyy automatisoimaan selaimen käyttöä. Seleniumin avulla selainta ohjataan toistamaan aina samat valinnat. Kun tiedosto on ladattu onnistuneesti selaimeen tulee latausaika näkyviin, jonka Selenium poimii talteen.

## Tiedostojen talletus

Tiedostojen talletukseen on toteutettu .NET-palvelin joka suorittaa talletusoperaatiot sekä Python ohjelma joka tekee pyynnön .NET-palvelimelle ja muodostaa tuloksista csv-tiedoston sekä kuvaajan.

# Projektien tulokset

Projektien tulokset ovat jaettu suorituskyvyn sekä kustannusten mukaan. Kustannusvertailussa otetaan kantaa talletusratkaisujen kustannuksiin Azure-pilvipalvelussa ja vertaillaan eri talletusratkaisujen kustannuksia. Tehokkuusvertailussa vertaillaan talletus- sekä lataustapoja ajallisessa näkökulmassa sekä lataustapojen käyttämää verkon määrää.

## Kustannusvertailu

Kustannusvertailussa vertaillaan Oracle Cloudin sekä Azuren tarjoamien talletusratkaisujen hintoja. Kustannusvertailussa käytetään yhden Teratavun kokeisen tallennustilan hintoja eri talletusratkaisuilla. Kustannusvertailu on esitetty taulukossa 1. Taulukosta 1 nähdään että saman talletusratkaisun hinta vaihtelee eri pilvialustoilla merkittävästi. Lisäksi eri talletusratkaisuilla on toisistaan eroavia hintoja.

|  |  |  |
| --- | --- | --- |
| palvelu | Oracle Cloud € /kuukausi | Azure € / kuukausi |
| Cosmos DB | - | 236 |
| Block Storage | 23,7 | 115,7 |
| Object Storage | 25.5 | 141 |
| File Storage | 279 | 128 |
| MYSQL | 67 | 118 |

## Tehokkuusvertailu

Kuva 11 esittää FTP-, FTPS- sekä SFTP-latausten latausaikojen tulokset Python- ja C#-kirjastoilla toteutettuna. Kuvasta nähdään että SFTP on selvästi FTPtä ja FTPStä hitaampi molemmilla ohjelmointikielillä suurimmilla tiedostoilla. Pienellä tiedostokoolla SFTP ei ole hitaampi kuin FTP tai FTPS, mutta ero kasvaa voimakkaasti kun tiedoston koko kasvaa. Python koodissa SFTP:n suorituskyky on selvästi huonompi, kuin mitä C#-koodilla on, mutta FTP- ja FTPS-latausten kohdalla ohjelmointiielellä ei ole merkitystä. FTP ja FTPS suorituskyvyt eroavat toisistaan vain vähän. Pienillä tiedostoilla FTP on hieman nopeampi koska aloitusprosessi on lyhyempi, koska yhteyttä ei salata.

Chart, line chart

Description automatically generated

Kuva 12 FTP, FTPS ja SFTP latausten tulokset

Kuva 12 esittää HTTP latauksien latausajat. Latausten pyynnön koko on esitetty x-akselilla ja latausaika on esitetty y-akselilla. Pylväiden väri puolestaan merkitsee eri pyynnön koodausta. Kuvaajasta näkee selvästi että pyynnön koon suurentuessa latausaika pienenee erittäin voimakkaasti kaikilla koodauksilla. Pienempi pyynnön koko aiheittaa useampia pyyntöjä palvelimelle, joka hidastaa latausta merkittävästi. 5 kB:n kokosilla pyynnöillä hitain koodaus on base64, vaikka teoriassa array on kaikista hitain koska se kasvattaa eniten siirrettävän datan kokoa.

Chart, bar chart

Description automatically generated

Kuva 13 HTTP-latausten latausajat

Kuva 13 esittä HTTP -latausten lähetettävän datan määrää suhteessa tiedoston kokoon. Lähetettävä data on mitattu Wireshark -ohjelmalla jota käytetään verkkojen analyysiin. Siirrettävässä datassa on mukana kaikki TCP- tasolla siirtnyt data, joten se katttaa siirretävän tiedosotn sekä HTTP- pyyntöön tarvittavan datan. Kun käytetään pientä 5kb kokoista pyyntöä lähetettävän datan määrä kasvaa reilusti kaikilla koodauksilla. Teoriassa form-data on kaikista tehokkain tapa lähettää dataa ja data siirtyy binäärimuodossa. Form-data kasvattaa lähetettävän datan määrää vain muutamalla prosentilla kun lähetettävän pyynnön koko on suuri. Base64 koodaus kasvattaa lähetettävän tiedoston kokoa noin 33% kun käytetään suurta pyyntöä. Array koodaus kasvattaa siirrettävän datan 100% suuremmaksi mitä alkuperäinen tiedosto, jonka takia se ehdottomasti huonoin tapa siirtää tiedostoja HTTP protokollan avulla. Ainoastaan 500 kB on nähtävissä selkeasti teoreettinen tulos että form-data on nopein jonka jälkeen base64 koodaus on noin 33% hitaampi sekä array useampi kertoja hitaampi.

Chart, bar chart

Description automatically generated

Kuva 14 HTTP -latausten tiedoston kasvu lähettäessä

Kuva 14 esittää Webocket-latauksen tulokset. Websocket-yhteys luotiin .NET:n SignalR-kirjaston avulla. SignalR-kirjasto on osa .NET joten sitä ei tarvitse erikseen asentaa palvelimelle, mutta selaimeen se tarvitsi asentaa npm-pakettina. Signal käyttää konepellin alla selaimen Websocket API:a, mutta se luo korkeamman tason rajapinnan, mahdollistaen palvelinpään funktioiden kutsumisen käyttöliittymästä websocket-yhteyden avulla. Binääriviestintää varten SignalR kirjastoon joutuu asentamaan lisäosan Nuget paketin nimeltään MessagePack sekä käyttöliittymään saman nimisen npm-paketin. Lataus suoritettiin sekä merkkijono viestinä että binäärinä ja testissä käytettiin kolmea erikokoista tiedostoa. Latausten ajat olivat hyvin samanlaiset riippumatta käytettiinkö merkkijono- vai binääriformaattia. Pienimmillä tiedostoilla merkkijono oli hitusen nopeampi, mutta suuremmalla tiedostolla binääri oli hieman nopeampi. Teoriassa binääriformaatti pitäisi olla nopeampi, koska siirrettävä data on pienempi.

Chart

Description automatically generated

Kuva 15 Websocket-lataus

Kuva 15 esittää siirrettävän datan suuruutta verrattuna alkuperäiseen tiedostoon. Siirrettävä data on mitattu TCP -tasolla, samaan tapaan kuin HTTP -pyyntöjen siirrettävän datan suuruutta vertailtaessa. String -formaatilla lähetettäessä siirrettävä data on kaksinkertainen alkuperäiseen tiedsotoon nähden ja binary -formattissa siirrettävä data 11% suurempi mitä alkuperäinen tiedosto.

Chart

Description automatically generated

Kuva 16 Websocket -latauksen siirrettävän datan kasvu

Kuva 15 esittää tiedostojen talletusnopeuksia eri talletusratkaisuilla. Kuvan y-akseli on logaritminen ja siitä erottuu selvästi etttä Disk ja File Storage ovat huomattavasti nopeimpia talletusratkaisuja. Tähän vaikuttavia tekijöitä on sisäverkon verkkoliikenne ja ne eivät vaadi mitään kirjautumismekanismia. SQL-talletus on seuraavaksi nopein. Siinä data siirretään TCP yhteydellä sisäverkossa, mutta SQL-kyselyt tarvitsevat kirjautumis- ja datan eheysprosessit jotka hidastavat tiedostojen tallettamista. Blob Storage sekä Cosmos käyttävät julkisen verkon HTTP API:a tiedostojen tallentamiseen ja ovat siksi ehdottomasti hitaimpia talletuspaikkoja. Blob Storage mahdollistaa binääritiedostojen tallentamisen binäärimuodossa, mutta Cosmos käyttää pelkästään json-dokumenttejä ja tiedostojen koko kasvaa merkittävästi kun on tallennettuna tekstinä. Cosmoksen json-dokumentin maksimikoko on 2MB joten Cosmoksen suorituskykyä ei voitu testata 10 MB tiedostoilla.

Chart, line chart

Description automatically generated

Kuva 17 Tiedostojen talletusnopeuksia

Yksi tutkimuksista on vertailla tiedostojen tallentamista SQL-kantaan binäärisenä blob tyyppina. Toinen mahdollinen vaihtoehto on tallentaa SQL-tietokantaan tiedoston tiedostopolku ja tallentaa tiedosto kovalevylle tietokannan sijaan. Kuva 16 esittää tutkimuksen tulokset. Tutkimuksessa vertailtiin tiedostojen lukua, kirjoitusta, poistoa sekä tiedosto infon lukua tietokannasta. Tutkimuksessa käytettiin 2 tietokanta taulua jotka ovat muuten identtiset, mutta toisessa tiedosto on tallennettuna tietokantaan ja toisessa linkin avulla tiedostojärjestelmään. Tiedostoinfo on tallennettuna samalla rivillä tiedoston tai sen linkin kanssa ja tyypiltään se on varchar. Tuloksista nähdään selvästi että tiedoston luku selvästi nopempi koska erillistä levylukuoperaatiota ei tarvitse tehdä tietokanta kyselyn jälkeen. Infon luku on hieman nopeampi taulussa jossa on linkki, johtuen että rivien indeksointi on hieman nopeampaa RAM muistissa koska rivien koko on pienempi. Kirjoitus- ja poistonopeudet ovat lähes samat riippumatta tallennustavasta. Toinen huomioita asia tiedostojen tallentamisessa levyjärjestelmään on tietokannan eheys. Kun tiedostot tallennetaan tietokantaan niin tietokanta pysyy eheänä tiedostojen suhteen. Levylle tallennettaessa eheys saattaa rikkoontua koska tietokantaoperaatio ja levyoperaatio eivät ole saman tietokantatransaktion sisällä

Chart, bar chart

Description automatically generated

Kuva 18 Tiedostojen tallentaminen SQL-tietokantaan ja kovalevylle

## Tietoturvavertailu

Tietoturvavertailussa vertaillaan eri talletustapojen sekä tiedonsiirtoprotokollien tietoturvallisuutta. Käsitellyistä tiedonsiirtotavoista suojaamattomia protokollia ovat HTTP ja FTP ja niiden käyttöä ei suositella nykyään olleenkaan, vaan niiden tilalla pitäisi käyttää suojattuja versioita HTTPS ja FTPS. Kun vertaillaan FTPS- ja SFTP-protokollia tietoturvan näkökulmasta, oleellisen asia on käytettävien porttien määrä. SFTP käyttää vain yhtä porttia tiedonsiirtoon, mutta FTPS tarvitsee kaksi. Lisäksi jokainen rinnakkainen yhteys tarvitsee oman porttinsa. Tämän takia FTPS -palvelimelle määritellään usein sallittujen porttien joukko palomuuriin. Palomuurit voidaan katsoa putkena, jotka määrittelevät sallitun ja kielletyn verkkotoiminnan yksityisessä verkossa. Palomuurit luovat tarkastuspisteitä verkkoliikenteen ohjaamiseksi, jolloin ne tarkistavat ohjelmoitujen parametrien perusteella verkkoliikennettä, ja toimivat niiden mukaisesti. Mitä vähemmän portteja on avattu, sitä vähemmän järjestelmässä on haavoittuvia kohtia. Jos käytössä on palomuuri, niin se pitää konffiguroida hyväksymään kaikki FTPS-palvelimen käyttämät portit, mikä lisää uhkia. Lisäksi salaus kiinnitetään FTPS -protokollaan TLS- tai SSL-tekniikoilla FTP -protokollan päälle. Tämän ulkoisen salauksen seurauksena palomuuriratkaisut eivät pysty havaitsemaan, mitä porttia käytetään ja miksi. Yksiporttiset SFTP-asetukset ovat ihanteellisia käytettäväksi palomuurin rinnalla. Se muodostaa yhden yhdistetyn yhteyden asiakkaan ja palvelimen välille. Palomuuri voi tarkkailla tämän yhteyden poikkeavuuksia, epäilyttäviä merkkejä ja muita uhkamerkkejä. [https://www.spiceworks.com/tech/networking/articles/sftp-vs-ftps/]

# yhteenveto

# Lähteet

1. <https://developer.mozilla.org/en-US/docs/Web/API/File>, viitattu 13.1.2021
2. <https://tools.ietf.org/html/rfc959>, viitattu 28.3.2021
3. <https://ieeexplore-ieee-org.libproxy.tuni.fi/stamp/stamp.jsp?tp=&arnumber=8068716>, viitattu 28.3.2021

Liite A: MS Wordin tekstityylien käyttö

Tyylien määrä pitää pyrkiä minimoimaan, jotta niitä on helppo käyttää ja muokata. Valitettavan monet kirjoittajat käyttävät Wordin tekstityylejä epäjohdonmukaisesti. Yleisesti ottaen tekstiä ei kuulu asemoida lisäämällä rivin- tai sivunvaihtoja, vaan nämä asiat tulee säätää tyyliasetuksista. Samoin kuvien, taulukoiden tai viitteiden numeroiminen käsin johtaa ongelmiin. Yleinen ongelmien lähde on tekstin kopioiminen muista dokumenteista. Usein teksti kannattaa tuoda ilman muotoiluja *Paste > Paste special > Unformatted text*, sillä muuten tyylien määrä räjähtää. Tekstin ulkoasun muokkaaminen näennäisen kätevästi Wordin yläreunan nappuloilla johtaa usein samankaltaisiin ongelmiin. Dokumentin sisällä tekstiä kopioidessa tai siirtäessä muotoilun voi toki säilyttää.

Avaamalla *Apply Styles*, näet käytetyt ja mahdolliset tekstityylit, missä niitä on käytetty ja voit muokata niitä tai luoda uusia. Valitsemalla vielä alareunasta *Options…* voit valita näytetäänkö todella käytetyt vai tässä dokumentissa mahdolliset. Lisäksi voi koiruuttaan valita kaikki Wordin tarjoamat, mutta tiettävästi kukaan ei ole vielä tarvinnut esimerkiksi 9. tason otsikoita. Lisäksi voit valita näytetäänkö pelkät kappaleasetukset (esim. *Normal*, *BibItem*…), fonttimuotoilut (*Normal + Italic*) ja listamuotoilut.

Toisinaan tekstin sekaan jää roikkumaan esimerkiksi lihavoituja kuvia, kursivoituja 10 pisteen rivinvaihtoja tai peräti punainen rivinvaihto jne. Jokaisesta listatusta tyylistä voi onneksi valita *Select all X instances*, jolloin näkee missä sitä on käytetty. Näin on helppo päästä eroon kummallisuuksista.. Esimerkiksi TTY:n opinnäyteohjeessa oli versiossa 11.3 (Kuva 1a) yli 130 tyyliä, mm. liki identtiset *Body text*, *Body text+Not italic, Normal*, *Plain text*, *After:6pt*, *After:6pt,Line spacing:Single*, *Before:1pt*, *Before:1pt,After:0pt*, *Before:3pt* ja niin edelleen. Tyylien määrä puolittui, kun nuo korvattiin yhdellä tyylillä *Normal*  ja muitakin turhia poistettiin (Kuva 1b).

Lisäksi voit valita muotoilumerkit näkyviin (engl. *Show paragraph marks and other hidden formatting symbols)* painamalla täytetyn P-kirjaimen peilikuvan näköistä ikonia yläpalkissa. Pedantit ihmiset voivat sitten poistaa tuplavälilyönnit, ylimääräiset rivinvaihdot, rivinvaihtoa edeltävät välilyönnit, ylimääräiset tabulaattorit ja muut roskat. Tarkista myös tekstin kopioinnista huolimatta kaikki lainausmerkit ovat samanlaiset, mielellään “…” eikä "…".

|  |
| --- |
|  |
| *a) Versio 11.3. Kiinnitä erityishuomio lukuisiin Arial-pohjaisiin tyyleihin, numeroituihin ja muihin listoihin sekä vasemmalle tasattuihin teksteihin.* |
|  |
| *b) Versio 11.5. Edelleen otsikkotasot 4-9 ovat tarpeettomia, mutta valitettavan vaikeita poistaa.*   1. Tyylit kirjoitusohjeen versioissa 11.3 ja 11.5 |

Jos dokumentin tulostaminen pdf-tiedostoon heikentää kuvien laatua, valitse asetus *High Quality Printing* eikä S*tandard*, tai käytä *File* > *Save as Adobe PDF*.

Muotoiluasetukset ovat muuttuneet Wordin versiossa 2013, mikä saattaa aiheuttaa ongelmia otsikoita ympäröivän tyhjän tilan kanssa ellei käytössä ole *Compatibility mode*, katso esimerkiksi https://answers.microsoft.com/en-us/office/forum/office\_2013\_release-word/where-can-i-find-suppress-extra-line-spacing-at/70bf7ca3-a884-40c4-ab59-34d2a04a1a8f