

Teemu Pöytäniemi

TIEDOSTOJEN SIIRTO JA TALLENNUS AZURE- PILVIPALVELUSSA

Diplomityö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Kari Systä
Tarkastaja: Toni Taipalus
10/2024

TIIVISTELMÄ

Teemu Pöytäniemi: Tiedostojen siirto ja tallennus Azure-pilvipalvelussa
Diplomityö
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
10/2024

Tässä diplomityössä tarkastellaan tiedostojen siirtämistä ja tallentamista Azure-pilvipalvelussa. Tiedostojen siirtämistä ja tallennusta analysoidaan kolmesta näkökulmasta: kustannukset, suorituskyky ja tietoturva. Tiedostojen siirtämistä vertaillaan eri tiedonsiirtoprotokollien avulla. Diplomityössä vertaillaan seuraavia suojaamattomia protokollia: HTTP, WebDav, WebSocket ja FTP. Lisäksi mukana ovat kaikkien suojaamattomien protokollien suojatut versiot sekä suojattu SFTP-protokolla. Tiedostojen tallennukseen käytetään SQL-tietokantaa, NoSQL-tietokantaa, Object storagea, Block storagea ja File storagea. Tietoturvan teoriaosuudessa käsitellään RSA- ja TLS-salausta.

Diplomityötä varten pystytettiin testausympäristö Azureen. Azuren virtuaalikoneeseen asennettiin SFTP-, FTPS- ja WebDav-palvelimet tiedonsiirtoa varten. HTTP- ja WebSocket-liikennettä varten pystytettiin .NET-palvelin. Angularilla tehdyllä käyttöliittymällä testattiin HTTP- ja WebSocket-liikennettä selaimesta. C#- ja Python-kirjastoja käytettiin tiedonsiirron testaamiseen eri protokollilla. Tiedostojen tallentamiseen käytettiin PostgreSQL-tietokantaa, Cosmos-tietokantaa, Azure Blob -tallennusta, Azure Files -tallennusta sekä Azure Managed Disks -tallennusta.

Kustannusvertailussa tarkastellaan Azuren tallennusratkaisujen hintoja kolmen eri tallennuskoon perusteella: 16 GB, 245 GB ja 1 TB. Työssä tutkitaan myös tallennusratkaisujen redundanssiasetusten vaikutusta hintaan. Azure tarjoaa kuumia tallennusratkaisuja, jotka tarjoavat hyvän suorituskyvyn mutta ovat kalliita, sekä kylmiä ratkaisuja, jotka ovat edullisia tallennustilan osalta, mutta suorituskyvyltään heikompia. Redundanssiasetuksilla, jotka määrittävät tiedostojen kopioiden säilytyksen, on merkittävä vaikutus hintaan. Mitä laajemmalle alueelle kopiot levitetään, sitä kalliimmaksi palvelu tulee.

Tiedostojen siirron kustannuksia vertaillaan epäsuorasti siirretyn internetliikenteen määrän perusteella. Jokaiselle protokollalle laskettiin, kuinka paljon ne kasvattavat tiedoston kokoa siirron aikana. SFTP-protokolla kasvatti tiedoston kokoa eniten ja muiden protokollien erot olivat erittäin pieniä.

Tehokkuusvertailussa tarkastellaan useita eri osa-alueita suorittamalla useita testejä. Tiedonsiirrossa havaittiin, että suojaamattomat protokollat ovat hieman nopeampia kuin suojatut, mutta erot olivat erittäin pieniä. Tiedostojen tallennuksessa huomattiin, että Cosmos-tietokanta ei sovellu binääritiedostojen tallentamiseen. Tallennusratkaisut, joita voitiin käyttää sisäverkossa, olivat huomattavasti nopeampia kuin ulkoverkossa olevat ratkaisut. Kuumien ja kylmien palvelumallien vertailussa havaittiin, että kaikilla tallennusratkaisulla oli yksi merkittävästi hitaampi palvelumalli, kun taas muiden palvelumallien suorituskyky oli lähes sama.

Tietoturva-vertailussa tutkittiin TLS-salausprotokollan tietoturvasuutta tiedonsiirrossa sekä vertailtiin eri siirtoprotokollien vaatimuksia palomuuriasetusten osalta. TLS-protokollan tietoturvaongelmat havaittiin liittyvän sen vanhentuneisiin versioihin, jotka ovat edelleen käytössä. Osiossa käytiin läpi myös Azuren pääsynhallinnan periaatteet sekä Azuren tallennusratkaisuihin ja pääsynhallinnassa havaitut tietoturvaongelmat. Lopuksi vertailtiin pilvipalveluihin liittyviä tietoturvatutkimuksia. Tutkimuksissa havaittiin, että suurin syy pilvipalveluissa kehitetyissä sovelluksissa ilmeneviin tietoturvaongelmiin liittyy inhimillisiin virheisiin, kun taas pilvipalveluiden tekniset tietoturvaongelmat ovat erittäin harvinaisia.

Tämän julkaisun alkuperäisyys on tarkastettu Turnitin Originality Check -ohjelmalla.

ABSTRACT

Teemu Pöytäniemi: File transfer and file storages on the Azure cloud platform
Master thesis
Tampere University
Master of engineering, information technology
10/2024

The thesis examines file transfer and file storage on the Azure cloud platform. File transfer and file storage are examined from three perspectives: cost, performance and security. File transfer is compared using different file transfer protocols. The thesis compares the following non-secure protocols: HTTP, WebDAV, WebSocket and FTP. In addition, secure versions of all non-secure protocols are included, as well as the secure SFTP protocol. SQL database, NOSQL database, Object Storage, Block Storage and File Storage are used to store the files. The theoretical part of the information security section deals with RSA and TLS encryption.

For the thesis, a testing environment was set up in Azure. SFTP, FTPS and WebDAV servers were set up on the Azure virtual machine for data transfer. A .NET server was set up for HTTP and WebSocket traffic. Angular front-end was used to test HTTP and WebSocket traffic from the browser. C# and Python libraries were used to test data transfer using different protocols. PostgreSQL Database, Cosmos Database, Azure Blob, Azure Files and Azure Managed Disks were used to store the files.

The cost comparison examines the prices of Azure storage solutions based on three different storage sizes: 16 GB, 245 GB and 1 TB. The study also examines the impact of redundancy settings on the price of storage solutions. Azure offers hot storage solutions, which offer good performance but are expensive, and cold storage solutions, which are inexpensive in terms of storage space but have lower performance. Redundancy settings, which determine how copies of files are stored, have a significant impact on price. The wider the area over which copies are distributed, the more expensive the service becomes.

The cost of transferring files is compared indirectly based on the amount of Internet traffic transferred. For each protocol, the increase in file size during transfer was calculated. The SFTP protocol increased the file size the most and the differences between the other protocols were very small.

A performance comparison compares several different things by carrying out several tests. In file transfer, it was found that the unprotected protocols were slightly faster than the protected protocols, but the differences were very small. For file storage, it was found that the Cosmos database is not at all suitable for storing binary files. The storage solutions that could be used on the internal network were significantly faster than the storage solutions on the external network. When comparing hot and cold service models, it was found that every storage solution had a significantly slower service model, while the performance of the other service models was almost the same.

The security comparison examined the security of the TLS encryption protocol for file transfer and compared the required firewall settings for different transfer protocols. Security problems with the TLS protocol were found to be related to outdated versions of the protocol that are still in use. The principles of Azure's access management were discussed in this section, as well as the security issues identified in Azure's storage and access management solutions. Finally, security studies on cloud platforms were compared. The studies found that the main cause of security problems in applications developed on cloud platforms is related to human error and that technical security problems in cloud platforms are very rare.

TEKOÄLYN KÄYTTÖ OPINNÄYTTEESSÄ

Opinnäytteessäni on käytetty tekoälysovelluksia:

Kyllä

Ilmoitukseni mukaan olen käyttänyt opinnäytteessäni tutkielmaprosessin aikana seuraavia tekoälysovelluksia:

Tekoälysovellusten nimet ja versiot: ChatGPT 4o.

Käyttötarkoitus: Kielioppivirheiden korjaus. ChatGPT on tarkastanut valmiin tekstin, ja on korjannut siitä kielioppivirheet ja parantanut lauserakenteita.

Osiot, joissa tekoälyä on käytetty: Koko työn viimeistelyyn.

Olen tietoinen siitä, että olen täysin vastuussa koko opinnäytteeni sisällöstä, mukaan lukien osat, joissa on hyödynnetty tekoälyä, ja hyväksyn vastuun mahdollisista eettisten ohjeiden rikkomuksista.

ALKUSANAT

Diplomityö on tehty mielenkiinnosta pilvipalveluita kohtaan, eikä se liity suoraan mihinkään yrityksen projektiin. Ajatus työn aiheesta syntyi yhdestä terveydenhuollon projektista, jossa aivokuvia ladataan pilvipalveluun.

Haluan kiittää Mika Välimäkeä arvokkaista neuvoista ja työn ohjauksesta.

Tampereella, 11.10.2024

Teemu Pöytäniemi

SISÄLLYSLUETTELO

1. JOHDANTO	1
2. TEORIA	3
2.1 Tiedostojen siirto	3
2.1.1 HTTP	3
2.1.2 WebDAV	6
2.1.3 WebSocket	7
2.1.4 FTP	10
2.1.5 SFTP	13
2.2 Tiedostojen tallennus	14
2.2.1 SQL-tietokannat	14
2.2.2 NoSQL	16
2.2.3 Object storage	18
2.2.4 Block storage	20
2.2.5 File storage	21
2.3 Pilvipalvelut	23
2.3.1 Microsoft Azure	23
2.4 Tietoturva	25
2.4.1 RSA	25
2.4.2 TLS	26
3. TESTAUSYMPÄRISTÖ	32
3.1 Tiedostojen siirto	33
3.2 Tiedostojen tallennus	35
4. PROJEKTIN TULOKSET	36
4.1 Kustannusvertailu	36
4.1.1 Tallennusratkaisuiden kustannukset	36
4.1.2 Tiedostojen siirron kustannukset	44
4.2 Tehokkuusvertailu	46
4.2.1 Tiedostojen siirto	46
4.2.2 Tiedostojen tallennus	56
4.3 Tietoturvavertailu	64
4.3.1 Suojaamattomat protokollat	65
4.3.2 TLS	66
4.3.3 Suojatut protokollat	69
4.3.4 Microsoft Azure	70
4.3.5 Pilvipalveluiden sovellusten tietoturvatutkimukset	73
5. YHTEENVETO	79
LÄHTEET	82

LYHENTEET JA MERKINNÄT

ADD	engl, Azure Active Directory, Microsoftin pääsynhallintajärjestelmä
AES	engl, Advanced Encryption Standard, standardoitu lohkosalaus
API	engl, Application Programming Interface, ohjelman rajapinta
ASCII	engl, American Standard Code for Information Interchange, 7-bittinen merkistökoodaus, joka sisältää ensisijaisesti amerikanenglannissa tarvittavat kirjaimet.
AWS	engl, Amazon Web Services, Amazonin pilvipalvelu
C#	ohjelmointikieli
DH	Diffie–Hellman, salausalgoritmi, jonka avulla voidaan jakaa avaimet symmetriseen salaukseen
DNS	engl, Domain Name System, palvelu IP-osoitteiden löytämiseen
EC2	engl, Elastic Compute Cloud, AWS:n virtuaalikonepalvelu
FCP	engl, Fibre Channel Protocol, verkkoprotokolla
FTP	engl, File Transfer Protocol, TCP:n päällä toimiva tiedonsiirtoprotokolla
FTPS	engl, File Transfer Protocol Secure, tiedonsiirtoprotokolla, joka on suojattu versio FTP-protokollasta
GRS	engl, Geo-redundant storage, Azuren redundanssiasetus, joka kopioi tiedoston kahdelle Azure-alueelle
HDD	engl, Hard Disk Drive, tallennusratkaisu, kiintolevy
HTTP	engl, Hypertext Transfer Protocol, tiedonsiirtoprotokolla, joka mahdollistaa internetin.”
HTTPS	engl, Hypertext Transfer Protocol Secure, tiedonsiirtoprotokolla, salattu versio HTTP-protokollasta
IDA	engl, Information Dispersal Algoritmia, tallennusalgoritmi, jota käytetään IBM:n Object storagessa
IO	engl, Input Output, sisään ja ulos menevä liikenne
IOPS	engl, Input Output Operations Per Second, IO- operaatioiden määrä sekunnissa
IP	engl, Internet Protocol, tiedonsiirtoprotokolla
JSON	engl, JavaScript Object Notation, dataformaatti
LRS	engl, Locally redundant storage, Azuren redundanssiasetus, joka kopioi tiedostot yhteen palvelinkeskukseen
NIST	engl, National Institute of Standards and Technology, Yhdysvaltain kansallinen standardisointi- ja teknologiainstituutti
NOSQL	engl, Not only SQL, tietokantamalli, joka ei käytä relaatiomallia
PCAP	engl, Packet Capture, verkkokäytäntö, jonka avulla voidaan analysoida verkkoliikennettä
QUIC	engl, general-purpose transport layer network protocol, Tiedonsiirtoprotokolla, jonka avulla HTTP/3 toimii
RC4	engl, Rivest Cipher 4, virtasalaus, jota käytetään SSL/TLS-salauksessa
REST	engl, Representational State Transfer, rajapinta-arkkitehtuuri
RSA	Rivest–Shamir–Adleman, julkisen avaimen salausjärjestelmä
SaaS	engl, software as a service, sovellus joka on tarjottu palveluna
SAN	engl, Storage Area Network, tietoverkko
SCSI	engl, Small Computer System Interface, kommunikointiprotokolla jonka avulla prosessori kommunikoi muistin kanssa
SFTP	engl, Secure File Transfer Protocol, tiedonsiirtoprotokolla, joka käyttää SSH-yhteyttä

SPA	engl, Single Page Application, selainkäyttöliittymä, joka tarjoillaan yhtenä sivuna
SSD	engl, Solid State Drive, tallennusratkaisu, kiintolevy
SSH	engl, Secure Shell, salattu tietoliikenneprotokolla
SSL	engl, Secure Sockets Layer, salausprotokolla, joka on vanhentunut
SQL	engl, Structured Query Language, kyselykieli ja tietokantamalli, joka käyttää relaatiomallia
TCP	engl, Transmission Control Protocol, tiedonsiirtoprotokolla joka toimii IP-kerroksen päällä
TLS	engl, Transport Layer Security, salausprotokolla, SSL-protokollan päivitetty versio
UDP	engl, User Datagram Protocol, tiedonsiirtoprotokolla, joka toimii IP-kerroksen päällä
URL	engl. Uniform Resource Locator, verkkosivun osoite
UTF-8	engl, Unicode Transformation Format-8, merkitöködaus
WAL	engl, Write Ahead Logging, tietokantojen protokolla
WebDav	engl, Web-based Distributed Authoring and Versioning, Tiedonsiirtoprotokolla, joka on HTTP-protokollan laajennus
XML	engl, Extensible Markup Language, rakenteellinen dataformaatti
XSS	engl, Cross-Site Scripting, hyökkäys uhrin selaimeen
ZRS	engl, Zone-redundant storage, Azuren redundanssiasetus, joka tallentaa tiedostot kolmeen eri käytettävyyssyöhykkeeseen

1. JOHDANTO

Moderni kaupallinen pilvilaskentainfrastruktuuri sai alkunsa vuonna 2002, kun Amazon Web Services (AWS) julkaistiin. Vuonna 2006 Amazon esitteli Simple Storage Service (S3) -palvelun ja Elastic Compute Cloud (EC2) -palvelun. Nämä palvelut olivat ensimmäisiä, jotka hyödynsivät palvelinvirtualisointia ja mahdollistivat asiakkaille palvelimien vuokraamisen heidän sovellustensa infrastruktuuriksi.[1]

Vaikka pilvipalveluiden kasvu oli aluksi hidasta, viimeisten 15 vuoden aikana ne ovat laajentuneet merkittävästi. Google, Microsoft, Oracle ja IBM julkaisivat omat pilvipalvelunsa vuosien 2010 ja 2013 välillä. Tämä auttoi tekemään pilvipalvelut yleisesti saataville, ja pilvipalveluiden käyttö lisääntyi merkittävästi. Nykyään pilvipalvelut ovat vallanneet suuren osan yritysmaailmasta, ja tuoreimpien tutkimusten mukaan yli 90 % yrityksistä käyttää pilvipalveluita.[2][3]

Pilvilaskenta-alustat voidaan jakaa kolmeen ryhmään: julkiseen, yksityiseen ja hybridiin. Julkiset alustat ovat yleisin pilvipalveluiden muoto. Niissä pilviresurssit (kuten palvelimet ja tallennustila) ovat kolmannen osapuolen pilvipalveluntarjoajan omistamia ja ylläpitämiä, ja ne toimitetaan internetin kautta. Microsoft Azure ja AWS ovat esimerkkejä julkisesta pilvestä. Julkisessa pilvessä samat laitteistot, tallennustilat ja verkkolaitteet jaetaan muiden organisaatioiden tai käyttäjien kanssa.[4]

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. Hybridipilvi taas yhdistää paikallisen infrastruktuurin tai yksityisen pilvipalvelun julkiseen pilvipalveluun. Hybridialustat mahdollistavat tietojen ja sovellusten liikkumisen kahden ympäristön välillä. Monet organisaatiot valitsevat hybridipilvilähestymistavan liiketoiminnan tarpeiden vuoksi, kuten säännösten ja tietosuojavaatimusten täyttämiseksi, paikan päällä tapahtuvan teknologiainvestoinnin hyödyntämiseksi tai alhaisen viiveen ongelmien ratkaisemiseksi.[4]

Tässä työssä tutkitaan pelkästään julkisia pilvipalveluita, erityisesti Microsoftin Azurea. Työssä tarkastellaan tiedostojen siirtämistä ja tallentamista Azure-pilvipalvelussa sekä vertaillaan julkisista pilvipalveluista tehtyjä tietoturvatutkimuksia. Tutkimuksissa on arvioitu, että julkisten pilvipalveluiden tallennusratkaisuiden liikevaihto vuonna 2023

olisi noin 99 biljoonaa Yhdysvaltain dollaria. Liikevaihdon ennuste vuodelle 2028 on 235 biljoonaa Yhdysvaltain dollaria. [5] Pilvipalveluiden tallennusratkaisuiden suuri kysyntä ja arviot kysynnän kasvusta olivat yksi motiivi työn tekemiselle. Työ rajoittuu pelkästään binäärimuotoisiin tiedostoihin, eikä se käsittele erikseen esimerkiksi tekstitiedostoja.

Työn teoriaosuudessa tiedostojen siirrosta esitellään kaikki yleisesti käytetyt protokollat, joiden avulla tiedostoja voidaan siirtää pilvipalveluihin. Tiedostojen tallennuksen osuudessa esitellään pääarkkitehtuurit, joita julkiset pilvipalvelut tarjoavat tallennusratkaisuille. Tietoturvaosuudessa käsitellään Rivest–Shamir–Adleman (RSA) -salausjärjestelmää sekä Transport Layer Security (TLS) -salausprotokollaa.

Työn tutkimusosassa käsitellään tiedostojen siirtoa ja tallennusta kolmesta eri näkökulmasta: kustannukset, suorituskyky ja tietoturva. Tutkimusta varten Azure-pilvipalveluun kehitettiin tarvittavat palvelut tutkittavien asioiden testausta varten. Azureen luotiin tiedonsiirtoprotokollille vaadittavat palvelimet, jotka integroitiin käytettäviin tallennusratkaisuihin. Tiedostojen siirrossa asiakasohjelmalla käytetään selaimessa toimivaa käyttöliittymää sekä ohjelmointikielien kirjastoja. Tiedostojen siirron ja tallennuksen suorituskykyä mitataan useilla erilaisilla suorituskykytesteillä. Tallennusratkaisuiden kustannuksia vertaillaan kolmen eri tallennuskokonaisuuden avulla. Työssä pohditaan, millaisia käyttötapauksia varten eri tallennusratkaisut on kehitetty, jotta niiden käyttäminen olisi mahdollisimman kustannustehokasta.

Tiedostojen siirron kustannuksia vertaillaan epäsuorasti siirretyn internetliikenteen määrän perusteella. Internetliikennettä tutkitaan WireShark-nimisen avoimen lähdekoodin pakettianalysaattorin avulla. Tietoturvaosuudessa esitellään suojaamattomien tiedonsiirtoprotokollien ongelmat ja tutkitaan TLS-salausprotokollan turvallisuutta. TLS-salausprotokollaa käytetään suojaamaan salaamattomia tiedonsiirtoprotokollia. Tässä osuudessa esitellään myös Azuren pääsynhallintamekanismit sekä tarkastellaan Azuren palveluiden tietoturva-avoittuvuuksia. Lopuksi vertaillaan muutamia tutkimuksia, jotka käsittelevät julkisille pilvipalveluille kehitettyjen sovellusten tietoturvaa.

2. TEORIA

2.1 Tiedostojen siirto

Tässä luvussa käsitellään protokollia, joiden avulla tiedostoja voidaan siirtää pilvipalveluihin.

2.1.1 HTTP

Hypertext Transfer Protocol (HTTP) on World Wide Webin (WWW) perusta, ja sitä käytetään verkkosivujen lataamiseen hypertekstilinkkien avulla. Hyperteksti on tekstiä, joka näytetään tietokoneen näytöllä tai muussa elektronisessa laitteessa, ja jossa on viittauksia hyperlinkeillä muihin teksteihin, joihin lukija pääsee välittömästi käsiksi. WWW käyttää Hypertext Markup Language (HTML) -tiedostoja määrittämään hypertekstitiedostojen rakenteen ja muotoilun. HTML on standardoitu merkintäkieli asiakirjoille, jotka on suunniteltu näytettäväksi verkkoselaimessa.[6]

HTTP on asiakas-palvelin-protokolla, jossa asiakasohjelma tekee palvelimelle pyynnön, johon palvelin vastaa. HTTP:n yleisin asiakasohjelma on verkkoselain, muita yleisiä asiakasohjelmia ovat mobiili- ja työpöytäsovellukset. Asiakasohjelma lähettää HTTP-pyyntöjä palvelimelle, joka vastaa siihen HTTP-vastauksella. HTTP-vastaus sisältää pyydetyt tiedot tai resurssit, kuten verkkosivun, kuvan tai videon. HTTP on tilaton protokolla, mikä tarkoittaa, että jokainen pyyntö ja vastaus ovat itsenäisiä, eivätkä ole riippuvaisia aiemmista pyynnöistä tai vastauksista.[7]

HTTP-pyyntö koostuu seuraavista osista: metodi, osoite, versio, otsikot ja runko. Metodi määrittää toiminnon, jonka asiakas haluaa suorittaa. Yleisimmät metodit ovat GET ja POST. GET-metodin avulla asiakas pystyy noutamaan tiedostoja palvelimelta, ja POST-metodin avulla asiakas voi lähettää dataa palvelimelle. Muita yleisiä metodeja ovat DELETE, PUT, PATCH, OPTIONS ja HEAD. osoite määrittää halutun resurssin sijainnin, ja sen määrittämisessä käytetään Uniform Resource Locator (URL) -osoitteita. Otsikot ovat avain-arvo-pareja tekstimuodossa, joiden avulla asiakas ja palvelin voivat välittää lisätietoja pyyntöjen ja vastausten mukana. Pyyntöjen runko sisältää tiedot, jotka asiakas lähettää palvelimelle, kuten kirjautumistiedot tai

palvelimelle siirrettävän tiedoston. Versio ilmaisee pyynnössä käytetyn HTTP-protokollan version.[8]

HTTP-vastaukset sisältävät tilakoodin, tilaviestin, version, otsikot ja rungon. Tilakoodi on kolminumeroinen luku, joka kertoo, onko tietty HTTP-pyyntö suoritettu onnistuneesti. Tilakoodit voidaan ryhmitellä viiteen kategoriaan pyynnön onnistumisen perusteella: informatiiviset vastaukset, onnistuneet vastaukset, uudelleenohjausviestit, asiakasvirheet ja palvelinvirheet. Tilaviesti on lyhyt tekstikuvaus tilakoodista. Versio, otsikot ja runko toimivat samalla tavalla kuin HTTP-pyyntössä.[8][9]

HTTP-protokollasta on julkaistu useita versioita: HTTP/0.9, HTTP/1.0, HTTP/1.1, HTTP/2 ja HTTP/3. HTTP-protokolla sai alkunsa 1980-luvun lopulla, ja ensimmäinen internet-palvelin kehitettiin CERNissä vuonna 1990. Vuonna 1991 julkaistiin ensimmäinen virallinen dokumentoitu HTTP-versio, nimeltään HTTP/0.9. Se tuki vain GET-pyyntöä, jonka avulla asiakkaat pystyivät noutamaan HTML-asiakirjoja palvelimelta, mutta ei muita tiedostomuotoja.[10][11]

HTTP/1.0 julkaistiin vuonna 1996, ja se käyttää yksinkertaista pyyntö-vastausmallia, jossa asiakas lähettää pyynnön palvelimelle ja palvelin vastaa asiakkaalle. Protokolla tukee GET-, POST- ja HEAD-pyyntöjä, ja siihen lisättiin otsikot sekä tilakoodit. Otsikoiden avulla protokolla mahdollistaa muidenkin tiedostotyyppien siirron kuin pelkkien HTML-dokumenttien. Pyyntö- ja vastausviestit lähetetään verkon yli tekstimuodossa, jossa otsikko ja teksti erotetaan tyhjällä rivillä. Jokaiselle pyynnölle luodaan oma TCP-yhteys, joka katkaistaan vastauksen saavuttua.[11][12]

HTTP/1.1 julkaistiin vuonna 1997, ja siitä julkaistiin päivitetty versio vuonna 1999. HTTP/1.1 tukee pysyviä yhteyksiä, joiden avulla useita pyyntöjä ja vastauksia voidaan lähettää peräkkäin saman TCP-yhteyden kautta. Tämä vähentää uusien yhteyksien luomistarvetta jokaiselle pyynnölle ja parantaa protokollan suorituskykyä. Yksi HTTP/1.1:n merkittävistä ominaisuuksista on HTTP-putkitus, jonka avulla useita HTTP-pyyntöjä voidaan lähettää saman TCP-yhteyden kautta ilman, että pyyntöjen vastauksia tarvitsee odottaa. Tämä parantaa huomattavasti HTML-sivujen latausaikoja, erityisesti korkean viiveen yhteyksissä. Palvelimen täytyy kuitenkin lähettää vastaukset samassa järjestyksessä kuin pyynnot vastaanotetaan.[11][12]

HTTP/1.1:n suurin suorituskykyongelma on head-of-line (HOL) -esto. Se tapahtuu, kun useampi pyyntö on menossa samaan kohteeseen, mutta jonon kärjessä oleva pyyntö, joka ei voi noutaa vaadittua resurssia, estää kaikki sen takana olevat pyynnot. HOL-esto ilmenee myös HTTP-putkessa, koska putkessa lähetettyihin pyyntöihin täytyy vastata samassa järjestyksessä. Rinnakkaisten TCP-yhteyksien lisääminen voi

helpottaa ongelmaa, mutta samanaikaisten TCP-yhteyksien määrä asiakkaan ja palvelimen välillä on rajallinen, ja jokainen uusi yhteys vaatii merkittäviä resursseja.[10][11][12]

HTTP/2 on päivitetty versio HTTP/1.1-protokollasta, joka otettiin käyttöön vuonna 2015. Se on suunniteltu korjaamaan joitakin HTTP/1.1:n rajoituksia ja suorituskykyongelmia sekä parantamaan verkkoviestinnän nopeutta ja tehokkuutta. Yksi merkittävimmistä eroista HTTP/1.1:n ja HTTP/2:n välillä on binäärikehystyskerros. HTTP/1.1 käsittelee kaikki pyynnöt ja vastaukset tekstimuodossa, kun taas HTTP/2 käyttää binäärikehystyskerrosta viestien kapselointiin binäärimuotoon säilyttäen silti HTTP-semantiikan. Viestien muuntaminen binäärimuotoon mahdollistaa HTTP/2:n käyttämät uudet tiedonsiirtotavat, joita ei ole saatavilla HTTP/1.1.[10][11]

HTTP/2:n binäärinen kehystyskerros koodaa pyynnöt ja vastaukset sekä pilkkoo ne pienemmiksi tietopaketeiksi, mikä lisää huomattavasti tiedonsiirron nopeutta. Toisin kuin HTTP/1.1, joka tarvitsee useita TCP-yhteyksiä HOL-eston vaikutuksen vähentämiseksi, HTTP/2 muodostaa yhden yhteyden kahden palvelimen välille, jossa on useita tietovirtoja. Jokainen tietovirta koostuu useista pyyntö- ja vastausviesteistä, jotka puolestaan jakaantuvat pienempiin yksiköihin, joita kutsutaan kehyksiksi. Yhteys koostuu joukosta binäärikoodattuja kehyksiä, joista jokainen on merkitty tiettyyn tietovirtaan. Tietovirran tunnisteet mahdollistavat kehysten lomittamisen siirron aikana ja niiden kokoamisen uudelleen toisessa päässä. Lomitetut pyynnöt ja vastaukset voivat toimia rinnakkain estämättä niiden takana olevia viestejä. Tätä prosessia kutsutaan multipleksaukseksi, ja se ratkaisee HTTP/1.1:n HOL-eston.[10][11]

Yksi mahdollinen ongelma on, että useat tietovirrat voivat olla riippuvaisia tietyn tietovirran valmistumisesta ja joutuvat odottamaan sen valmistumista. Ongelman ratkaisemiseksi käytetään tietovirtojen priorisointia, jonka avulla tietovirralla voidaan määrittää prioriteetti ja riippuvuussuhde muihin tietovirtoihin. Palvelin käyttää näitä tietoja luodakseen riippuvuuspuun, jonka avulla se voi määrittää, missä järjestyksessä tietovirtojen kehykset lähetetään.[10][11]

HTTP:n uusin versio, HTTP/3, on Googlen kehittämä, ja se julkaistiin vuonna 2015. Suurin muutos verrattuna HTTP/2:een on taustalla olevan siirtokerroksen verkkoprotokollan TCP:n vaihtaminen Quick UDP Internet Connections (QUIC) -protokollaan. QUIC on Googlen vuonna 2012 kehittämä protokolla, joka käyttää siirtokerroksessa UDP:ta TCP:n sijaan. QUIC on multipleksoitu viestintäalgoritmi, joka kehitettiin tehokkaammaksi kuin HTTP/2:n käyttämä multipleksaus.[13][14]

Vaikka HTTP/2 poistaa HOL-eston HTTP-protokollasta, TCP-protokollassa on oma HOL-esto. Jos TCP hukkaa paketin yhdessä HTTP/2:n luomassa tietovirrassa, kaikkien muiden tietovirtojen kehysten siirtäminen samassa TCP-yhteydessä keskeytyy, kunnes hukatut paketit on lähetetty uudelleen. QUIC puolestaan jakaa tietovirran datan pienempiin osiin, joita kutsutaan kehyksiksi. Kehykset lähetetään UDP-paketteina, ja ne sisältävät tietovirran tunnuksen ja järjestysnumeron. Vastaanottaja voi järjestää ne uudelleen, jos ne vastaanotetaan eri järjestyksessä kuin ne lähetettiin. Jos paketti katoaa, QUIC lähettää sen uudelleen aikakatkaisun jälkeen. Logiikka on sama kuin TCP:llä HTTP/2-protokollassa, mutta QUIC:n toiminta yhdessä tietovirrassa ei vaikuta yhteyden muihin tietovirtoihin.[13][14]

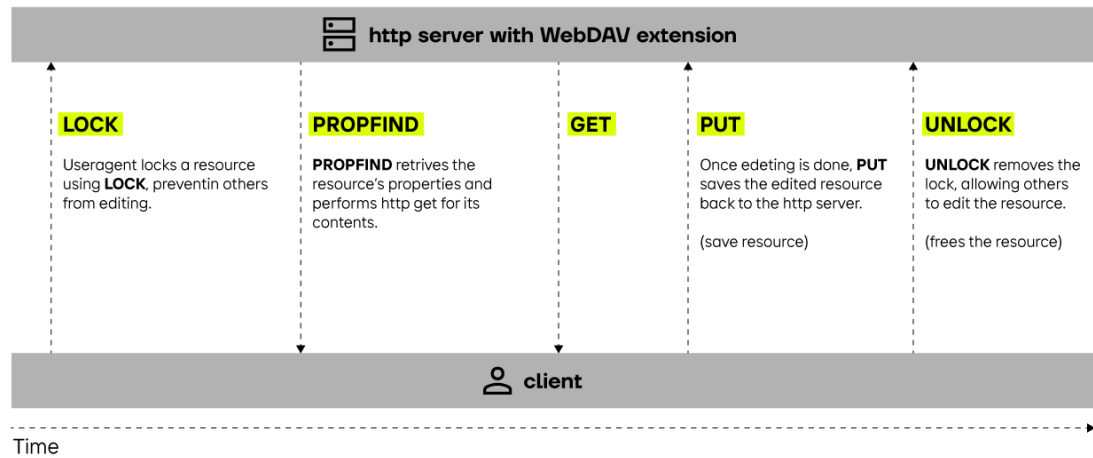
2.1.2 WebDAV

Web Distributed Authoring and Versioning (WebDAV) on HTTP/1.1:n laajennus. WebDAV-protokolla esiteltiin alun perin vuonna 1996, ja se standardoitiin ensimmäistä kertaa vuonna 1999. Protokollan nykyinen versio on vuodelta 2007. WebDAV mahdollistaa HTTP-palvelimen toimimisen tiedostopalvelimena, mahdollistaen useamman asiakkaan työskentelyn saman tiedoston parissa samanaikaisesti. WebDAV laajentaa HTTP-otsikoiden ja -metodien standardijoukkoa. WebDAV:n avulla voi luoda, siirtää, muokata, poistaa ja kopioida tiedostoja ja kansioita.[15][16][17]

WebDAV-palvelimet mahdollistavat tiedostojen versioiden seurannan, ja ne on jaettu kahteen luokkaan versioiden seurannan perusteella: luokka 1 ja luokka 2. Luokan 1 WebDAV-palvelimet tarjoavat perushallintaominaisuuksia, kuten mahdollisuuden luoda, kopioida, siirtää tai poistaa tiedostoja ja kansioita. Monet asiakasohjelmat pitävät luokan 1 WebDAV-palvelimia vain lukuoikeuksilla, koska ne eivät voi suojata tiedostoja samanaikaisilta muutoksilta. Luokan 2 WebDAV-palvelimet voivat lukita tiedostoja ja sallivat niiden samanaikaisen muokkauksen.[15][16][17]

WebDAV lisää seuraavat HTTP-metodit HTTP/1.1-standardiin: PROPFIND, PROPPATCH, COPY, MOVE, MKCOL, LOCK ja UNLOCK. WebDAV hyödyntää myös PUT- ja DELETE-metodeja, jotka kuuluvat jo HTTP-standardiin. Protokolla viittaa tiedostoihin ja kansioihin resursseina. PROPFIND-metodilla voi hakea tietoja resursseista, ja PROPPATCH-metodi mahdollistaa niiden muokkaamisen. LOCK-metodi lukitsee valitun resurssin niin, että muut käyttäjät eivät voi tehdä siihen muutoksia, ja UNLOCK-metodi purkaa lukituksen. COPY-metodi kopioi resurssin palvelimella, ja MOVE-metodi siirtää sen. Resurssin voi ladata palvelimelle PUT-metodilla, poistaa DELETE-metodilla ja ladata GET-metodilla.[17]

Kuva 1 esittää esimerkitapauksen asiakkaan ja palvelimen välisestä liikenteestä, jossa asiakas ensin lukitsee tiedoston, muokkaa sitä ja lopuksi purkaa lukituksen.[18]

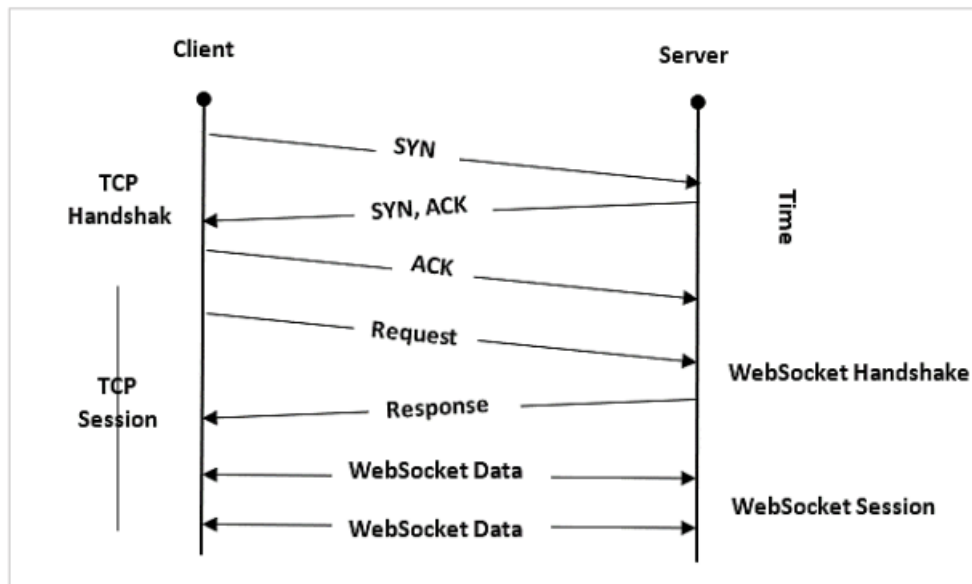


Kuva 1. Esimerkki WebDav-latauksesta.[18]

2.1.3 WebSocket

WebSocket-protokolla luo kaksisuuntaisen reaaliaikaisen yhteyden asiakkaan ja palvelimen välille. Tämä mahdollistaa sen, että palvelin voi lähettää dataa asiakkaalle ilman asiakkaan erillistä pyyntöä. WebSocket on tilallinen protokolla, mikä tarkoittaa, että asiakkaan ja palvelimen välinen yhteys pysyy aktiivisena, kunnes jompikumpi osapuoli lopettaa sen. Kun asiakas tai palvelin sulkee yhteyden, yhteys katkeaa molemmista päistä. WebSocket toimii TCP:n yli toimivana siirtokerrosprotokollana. Vaikka WebSocket muistuttaa TCP-protokollaa, ne eroavat toisistaan, eikä WebSocketilla voi muodostaa yhteyttä TCP-palvelimeen. WebSocketin pääasiallinen asiakasohjelma on selain, mutta myös muilla asiakasohjelmilla voidaan luoda yhteys palvelimeen. WebSocket API on selaimen ohjelmointirajapinta, joka mahdollistaa yhteyden muodostamisen selaimesta WebSocket-palvelimeen JavaScript-ohjelmointikielen avulla.[19][20]

Kuva 2 esittää WebSocketin arkkitehtuurin korkealla tasolla. Aluksi kättelyvaiheessa käytetään HTTP-protokollaa, minkä jälkeen siirrytään WebSocket-protokollaan.



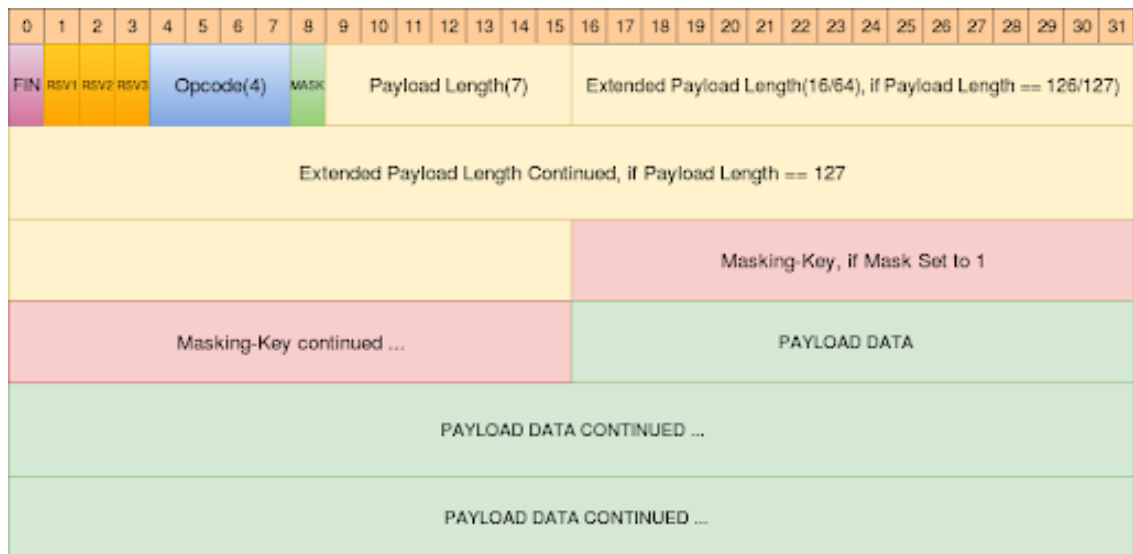
Kuva 2. WebSocket-protokolla.[21]

Asiakas aloittaa kättelyprotokollan lähettämällä HTTP GET -pyynnön palvelimelle. Pyynnössä tulee olla mukana tietyt HTTP-otsikot, jotta palvelin hyväksyy pyynnön. Upgrade- ja Connection-otsikoiden arvoilla kerrotaan palvelimelle, että yhteys halutaan vaihtaa WebSocket-protokollaan. WebSocketin version tulee olla 13, joka on nykyinen versio WebSocket-standardista. Pyynnössä on myös mukana Sec-WebSocket-Key-otsikko, jonka arvo on satunnainen 16-tavuinen merkkijono, joka on base64-koodattu. Jos palvelin hyväksyy kättelyn, HTTP-vastauskoodi on 101.[20][22]

Kun kättely on onnistuneesti suoritettu, siirrytään WebSocket-protokollaan. Kuva 3 esittää protokollassa siirtyviä kehyksiä, joiden avulla data kulkee. Kehyksen lohkoilla on seuraavat merkitykset alkaen ensimmäisestä bitistä:

- **FIN:** Tämä yksi bitti ilmaisee, onko kehys viestin viimeinen osa vai onko tulossa lisää kehyksiä.
- **RSV1, RSV2, RSV3:** Nämä kolme bittiä on varattu tulevia laajennuksia varten. Niiden tulee olla nollia, ellei jokin laajennus ole asettanut niitä muuksi.
- **Opcode:** 4-bittinen kenttä, joka ilmoittaa kehyksen tyyppin, kuten tekstin, binääridatan tai ohjausviestin.
- **Mask:** Tämä yksi bitti ilmaisee, onko viesti naamioitu vai ei. Asiakkaan lähettämät viestit tulee aina olla naamioitu.
- **Datan-pituus:** Tämä kenttä ilmaisee datan pituuden
- **Naamiointiavain:** Jos Mask-bitti on asetettu, tämä 32-bittinen arvo sisältää maskin, joka käytetään datan naamioimiseen.

- **Siirrettävä data:** Tämä lohko sisältää itse siirrettävän datan, kuten tekstin tai binääridatan.



Kuva 3. WebSocket-kehys.

Kun asiakas lähettää kehyksen palvelimelle, MASK-bittillä täytyy olla arvo 1, ja 32-bittinen naamiointiavain tulee olla määriteltynä. Tämän avaimen on oltava jokaiselle kehykselle erilainen, ja sen tulee olla luotu kryptografisesti vahvalla algoritmilla, jotta tulevia arvoja ei voida ennustaa aiempien perusteella. Datan naamiointi tapahtuu 8 bitin lohkoissa, ja prosessissa käytetään jakojäännös- ja XOR-operaatioita.[20]

Datan naamioimisen tarkoituksena on tehdä WebSocket-liikenteestä erottuvaa ja arvaamatonta verrattuna HTTP-liikenteeseen. Ilman tätä verkkoinfrastruktuurilaitteet, jotka eivät ole päivitetty käsittelemään WebSocket-liikennettä, saattaisivat virheellisesti luulla sitä normaaliksi HTTP-liikenteeksi. Tämä voisi avata tietoturva-aukkoja. Naamioimalla data voidaan estää tällaisten laitteiden mahdollisesti aiheuttamat riskit. Vaikka hyökkääjä pystyisi hallitsemaan sekä asiakas- että palvelinpuolen koodia, hän ei pysty manipuloimaan kehyksen arvoja, koska naamiointi tekee niistä satunnaisia ja siten vaikeasti ennustettavia. Tämä parantaa verkkoyhteyden tietoturvaa huomattavasti.[23]

4-bittisellä Opcode-koodilla pystytään merkitsemään kehyksen tyyppi. Kehys voi olla datan siirtoon tarkoitettu kehys tai hallintakehys. Jos kehys on hallintakehys, sillä on sovittu hallintakoodi Opcode-lohkoissa ja mahdollisesti siirrettävää dataa. Mahdollisia komentoja ovat 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,

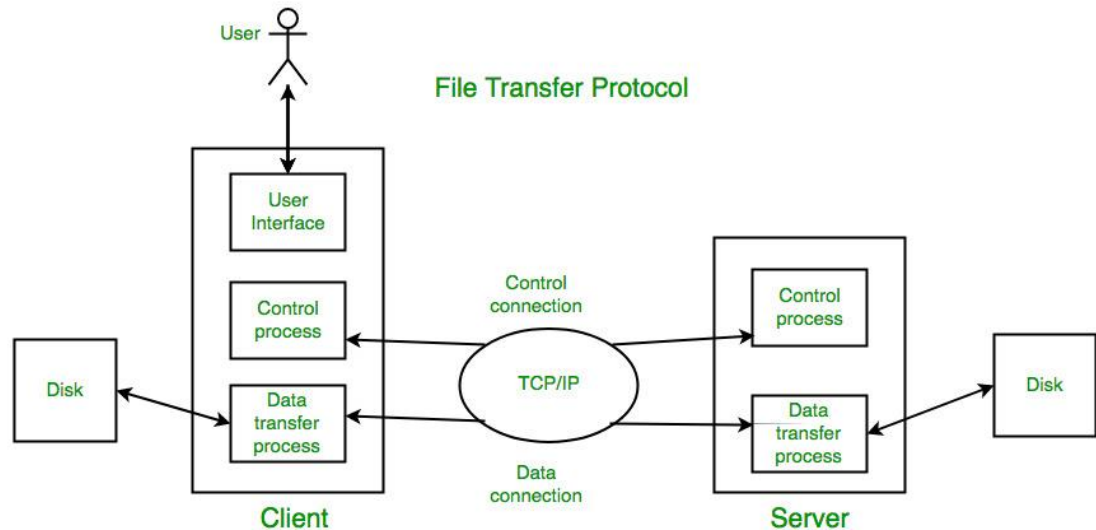
jonka komento on myös close. Close-vastausta voi viivästyttää ja jatkaa datan lähettämistä, mutta silloin ei ole takuita siitä, että close-komennon lähettänyt osapuoli pitää yhteyden auki. Ping-komennon tarkoitus on tarkistaa, että yhteyden toinen pää vastaanottaa edelleen liikennettä. Kun ping-komento on vastaanotettu, siihen vastataan lähettämällä pong-komento.[20][23]

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 kehysten datan formaatin on oltava sama kuin ensimmäisen kehyksen. Ensimmäinen kehys määrittää Opcode-arvolla siirrettävän datan tyytin, ja seuraavien kehysten Opcode määrittää, että ne ovat jatkoa edelliselle kehykselle. Datan viimeisen kehyksen FIN-bitin arvo on 1, mikä kertoo, että kyseessä on datan viimeinen kehys.[20][23]

Opcode-koodissa on varattuja komentoja lisäosia varten, joita ei ole määritelty WebSocket-protokollassa. Asiakas voi pyytää lisäosan käyttöä kättelyvaiheessa lisäämällä Sec-WebSocket-Extensions-otsikon pyyntöön. Jos serveri hyväksyy lisäosien käytön, se lisää saman Sec-WebSocket-Extensions-otsikon vastaukseen, johon se listaa hyväksytyt lisäosat.[20][23]

2.1.4 FTP

File Transfer Protocol (FTP) on tiedostojen siirtoon käytetty protokolla. Protokolla toimii TCP/IP-yhteyden päällä, ja se koostuu asiakkaasta ja palvelimesta. Kuva 4 esittää 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.[24]



Kuva 4. FTP-protokolla.[25]

Protokolla käyttää kahta rinnakkaista TCP-yhteyttä: hallintayhteyttä ja datayhteyttä. Hallintayhteydellä lähetetään kaikki komennot, jotka liittyvät käyttäjien hallintaan ja tiedonsiirtoon, ja se on toteutettu Telnet-protokollalla. Telnet on vanha sanomamuotoinen protokolla, joka toimii TCP-protokollan päällä. Telnetillä pystyy muodostamaan yhteyden etätietokoneelle, ja sitä on aikoinaan käytetty muun muassa sähköpostien lähetykseen. Telnet ei salaa sanomiaan, minkä vuoksi protokolla on tietoturvaltaan heikko. Telnetin käytöstä on luovuttu moderneissa järjestelmissä, ja se on korvattu turvallisemmilla protokollilla, kuten SSH:lla. Hallintayhteydellä määritellään datayhteyden ominaisuudet ja sillä avataan haluttu datayhteys.[24]

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 muutamia eri tiedonsiirtotyypppejä, joilla se pystyy esittämään siirrettävän datan. Oletustyyppi on ASCII, jota kaikkien FTP-toteutusten täytyy tukea. ASCII on tekstin merkistö, jossa jokainen merkki esitetään 7-bittisenä numerona. Tiedosto lähetetään 8-bittisenä NVT ASCII:na, jossa eniten merkitsevä bitti on nolla ja sen perässä on 7-bittinen ASCII-koodi. Vaihtoehtoinen tiedonsiirtotyyppi tekstitiedostoille on EBCDIC. Se on 8-bittinen merkistö, jota käytetään pääasiassa IBM:n suurtietokoneissa. ASCII- ja EBCDIC-tyypit voivat ottaa myös toisen vapaaehtoisen parametrin, joka määrittää tekstin pystysuuntaista asettelua ja sitä käytetään tekstin tulostuksen muotoiluun.[24]

Image-tyyppi on tarkoitettu binääritiedostojen lähettämiseen. Tiedosto luetaan biteiksi ja bitit lähetetään 8-bittisinä tavuina. Lähetyksen loppuun saatetaan tarvita täytettä, jos tiedoston bitit eivät ole jaollisia kahdeksalla. 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 annetulla parametrilla.[24]

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.[24]

Datan siirtämisen aloittaminen edellyttää datayhteyden muodostamista sekä siirtoparametrien sopimista. Ensin asiakkaan on avattava datayhteyden portti kuuntelua varten ja lähetettävä pyyntösanoma hallintayhteydellä. Tämä sanoma sisältää tiedon siitä, mihin suuntaan dataa ollaan siirtämässä. Palvelin vastaa pyyntöön ottamalla yhteyden asiakkaan datayhteyden porttiin. Seuraavaksi sovitaan tiedoston siirtotapa. FTP tukee kolmea siirtotapaa: stream, block ja compressed.[24]

Stream-tilassa tiedostot siirretään jatkuvana virtana. Jos tiedoston rakenne on tiedostomuotoinen, tiedoston päättymisen merkitään sulkemalla datayhteys. Tallennerakenteessa käytetään kaksitavuista hallintakoodia osoittamaan osien tai koko tiedoston päättymistä.[24]

Block-siirrossa data jaetaan lohkoihin, ja jokaisella loholla on 3-tavuinen otsikko, joka sisältää tiedot lohkon pituudesta ja tyypistä. Lohkon tyypin avulla voidaan määrittellä, onko kyseessä tiedoston viimeinen lohko, tallenteen viimeinen lohko vai uudelleenkäynnistysmerkki.[24]

Compressed-siirrossa käytetään yksinkertaista pakkausalgoritmia, nimeltään run-length encoding, joka lähettää dataa lohkoina samalla tavalla kuin block-siirrossa. Pakkaus vähentää siirrettävän datan määrää, mutta yleensä tiedostot pakataan tehokkaammilla algoritmeilla jo niiden luontivaiheessa tai verkkotasolla, esimerkiksi modeemien toimesta.[24]

FTP-protokolla ei sisällä sisäänrakennettua tapaa tarkistaa siirrettävän datan eheyttä, vaan se luottaa TCP-protokollaan tässä tehtävässä. FTP tukee kuitenkin tiedostonsiirron jatkamista, jos lataus keskeytyy. Latauksen jatkaminen on mahdollista vain, jos siirtotapana on block tai compressed. Jatkaminen merkitään lohkon otsikkoon varatulla koodilla, joka osoittaa, mistä tiedoston lataus jatkuu.[24]

FTP-protokolla ei salaa kumpaakaan käyttämäänsä yhteyttä, joten sitä pidetään nykyään tietoturvattomana. FTPS (FTP Secure) on FTP-protokollan turvallisempi lisäosa, joka käyttää SSL- tai TLS-suojausprotokollaa. Suojauksen käyttöönotto voidaan toteuttaa kahdella eri tavalla: implisiittisesti ja eksplisiittisesti. Implisiittinen tapa edellyttää suojasta heti yhteyden muodostamisvaiheessa. Tämä rikkoo yhteensopivuuden pelkästään FTP-protokollaa tukevien asiakasohjelmien ja palvelimien kanssa, jotka eivät tue implisiittistä suojasta. Eksplisiittisessä suojauksessa asiakas pyytää erikseen suojattua yhteyttä. Tämä mahdollistaa sen, että palvelin voi tukea sekä FTP- että FTPS-asiakkaita. Eksplisiittisessä suojauksessa voidaan lisäksi määritellä, mitkä yhteydet suojataan, kun taas implisiittisessä suojauksessa molemmat kanavat (hallinta- ja datayhteys) ovat suojattuja. Esimerkiksi, jos siirrettävä data on jo valmiiksi salattu, voidaan suojata vain hallintayhteys, kun taas datayhteyttä ei välttämättä tarvitse suojata.[24][26]

2.1.5 SFTP

Secure File Transfer Protocol (SFTP) on tiedostonsiirto- ja tiedostonhallintaprotokolla, joka koostuu palvelimesta ja asiakasohjelmasta. SFTP toimii suojatun kanavan kautta, ja se suunniteltiin SSH-2-protokollan laajennukseksi. SFTP ei suojaa liikennettä itsessään, vaan se luottaa alapuolella olevan kanavan, kuten SSH:n, suojaukseen. SFTP-yhteyden avulla voidaan siirtää tiedostoja sekä suorittaa tiedostonhallintaan liittyviä toimintoja. Vaikka SFTP suunniteltiin alun perin SSH-2-protokollan laajennukseksi, sitä voidaan käyttää myös muiden protokollien, kuten SSH-1:n, kanssa. SSH-1-protokollan käyttö aiheuttaa kuitenkin rajoituksia asiakasohjelmalle, sillä sen on tiedettävä SFTP-palvelimen binääritiedostojen tarkka sijainti palvelimella.[27][28]

SFTP-protokollasta on julkaistu useita versioita, joista nykyisin käytetyin on versio 3, joka julkaistiin vuonna 2001. Tämän jälkeen on julkaistu versiot 4, 5 ja 6, mutta ne eivät ole saaneet laajaa tukea. Useat SFTP-palvelimien kehittäjät ovat ilmoittaneet, että heillä ei ole aikomusta tukea uusimpia versioita tulevaisuudessa.[27][28]

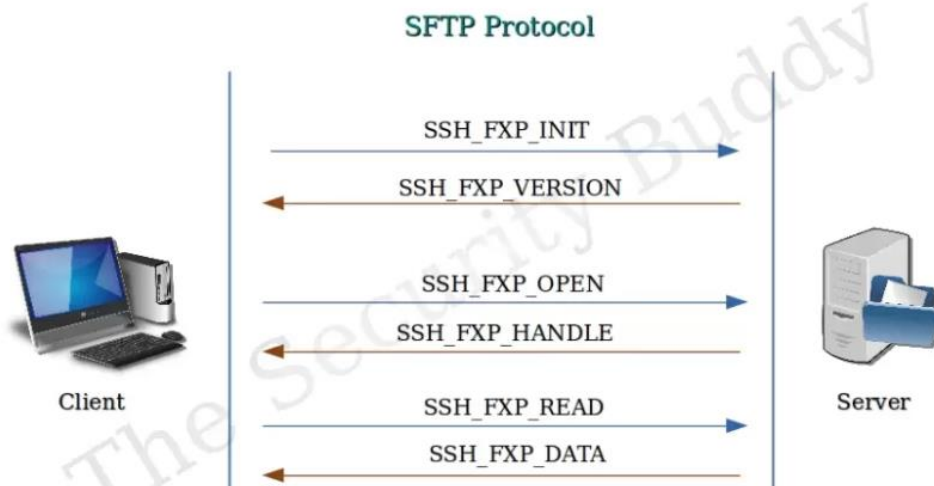
SFTP-protokollan viestit siirretään SSH-protokollan kautta suojatussa kanavassa. SFTP-viestit koostuvat erilaisista viestityypeistä, joilla on eri tehtäviä, kuten tiedostojen avaaminen, luku- ja kirjoitusoperaatiot sekä virheiden käsittely. Jokainen SFTP-protokollan viesti koostuu seuraavista pääosista: [28]

- **Pituuskenttä:** 32-bittinen kokonaisluku, joka ilmaisee viestin kokonaispituuden tavuina.

- **Viestin tyyppi:** 8-bittinen kokonaisluku, joka kertoo viestin tyyppin. Viestin tyyppi määrittää, millaisia tietoja viesti sisältää. Esimerkkejä viestityypeistä ovat aloitusviesti SSH_FXP_INIT, tiedoston avausviesti SSH_FXP_OPEN ja tiedoston lukuviesti SSH_FXP_READ.
- **Viestin sisältö:** Tämä kenttä vaihtelee viestin tyyppin mukaan ja sisältää viestin varsinaisen sisällön.

Kuva 5 esittää esimerkin SFTP-protokollan käytöstä, jossa näkyvät viestien tyyppien nimet. Asiakasohjelma lähettää aluksi SSH_FXP_INIT-komennon, jonka mukana on haluttu protokollan versio. Palvelin vastaa SSH_FXP_VERSION-komennolla, joka sisältää palvelimen tukemat versiot. Asiakasohjelman ja palvelimen on tuettava samaa protokollaversiota, jotta yhteys voidaan muodostaa.[28]

Tiedostojen käsittely tapahtuu niille tarkoitetuilla komennoilla. Esimerkiksi SSH_FXP_OPEN-komento avaa tiedoston tai luo uuden tyhjän tiedoston. Palvelin vastaa SSH_FXP_OPEN-kyselyyn SSH_FXP_HANDLE-komennolla, jos tiedoston avaaminen onnistuu. SSH_FXP_HANDLE palauttaa datana ID:n, jonka avulla asiakasohjelma voi käyttää tiedostoa. Asiakas voi tämän jälkeen lukea tiedoston sisällön SSH_FXP_READ-lukukomennolla.[28]



Kuva 5. SFTP-protokolla.[29]

2.2 Tiedostojen tallennus

Tässä luvussa käsitellään eri tallennusratkaisuita, joita pilvipalvelut yleisesti tarjoavat.

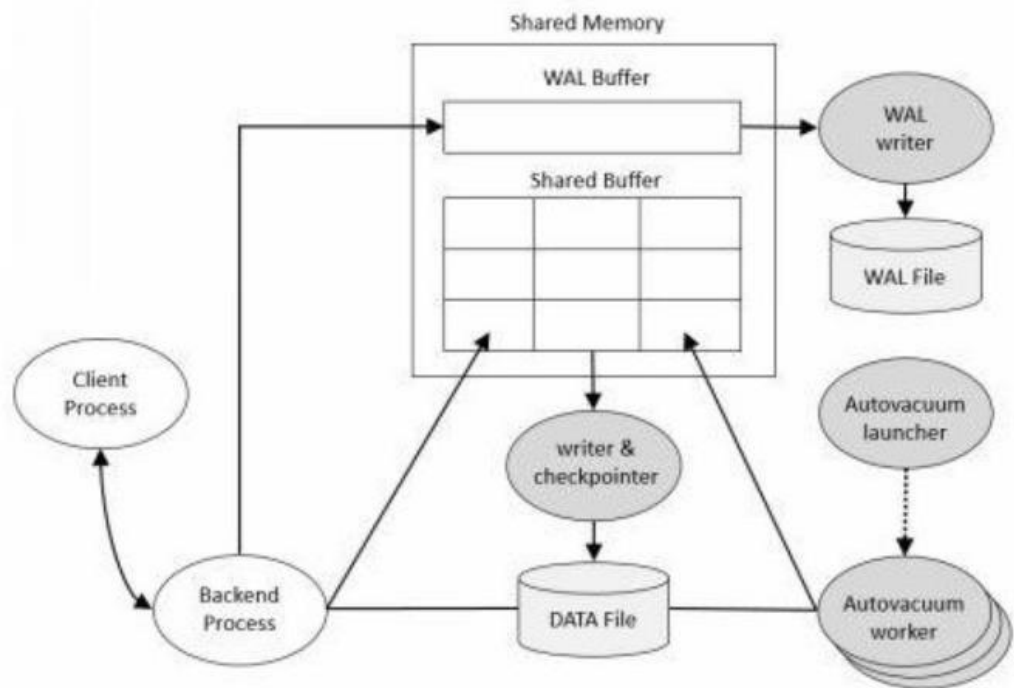
2.2.1 SQL-tietokannat

SQL-tietokannat ovat relaatiotietokantoja, joissa tieto on tallennettuna tauluihin, jotka koostuvat riveistä ja sarakkeista. Taulut perustuvat relaatiomalliin, joka pohjautuu predikaattilogiikkaan. Taulu esittää tiettyyn aiheeseen liittyviä tietoja, sarakkeet kuvaavat kokonaisuuden eri ominaisuuksia, ja rivit edustavat kyseisen asian yksittäisiä ilmentymiä. Taulujen riveillä on uniikki tunniste, eli ID, joka voi koostua yhdestä tai useammasta sarakkeen arvosta. Taulujen välille voidaan luoda yhteyksiä avainten avulla. Avaimet toimivat viitteinä toisten taulujen arvoihin, mahdollistaen tietojen yhdistämisen ja viittaamisen eri taulujen välillä.[30]

Tietokannan käyttöä mahdollistavat tietokannan hallintajärjestelmät (DBMS, Database Management Systems). Ne mahdollistavat käyttäjien pääsyn tietokantaan sekä sen päivittämisen ja muokkaamisen. Tietokannan hallintajärjestelmiä on sekä maksullisia että avoimen lähdekoodin ilmaisia vaihtoehtoja. Suosituimmilla avoimen lähdekoodin hallintajärjestelmillä on hyvin samankaltainen pääarkkitehtuuri, vaikka niiden toteutuksissa saattaa olla pieniä eroavaisuuksia.[30]

Kuva 6 esittää PostgreSQL-tietokannan arkkitehtuuria. PostgreSQL on yksi suosituimmista avoimen lähdekoodin tietokannoista, ja se julkaistiin vuonna 1995. PostgreSQL toimii asiakas-palvelin-mallilla, jossa asiakasohjelma ottaa yhteyden hallintajärjestelmään. PostgreSQL toimii prosessimallilla, jossa jokaiselle yhteydelle luodaan erillinen prosessi. Tämä malli eroaa monista muista tietokannoista, jotka käyttävät säie- tai tapahtumapohjaista mallia. Kaikilla prosesseilla on pääsy jaettuun muistiin, joka sijaitsee RAM-muistissa. Tietokantakyselyt kohdistuvat jaettuun muistiin Shared Buffers -muistialueelle, johon luetut tiedot tauluista ja indekseistä tallennetaan. Jos pyyntö koskee tietoja, jotka ovat jo Shared Buffers -muistissa, tietokanta voi palauttaa tiedot ilman levyoperaatiota, mikä nopeuttaa suorituskyyä merkittävästi.[31][32][33]

Kirjoitusoperaatiot käsitellään ensin Write-Ahead Log (WAL) -logissa. Muutokset tehdään ensin jaettuun muistiin, ja sen jälkeen WAL-kirjoittaja kirjoittaa muutokset levylle säännöllisenä taustaprosessina. Checkpointer- ja writer-prosessit päivittävät muutokset jaetusta muistista levylle sen jälkeen, kun muutokset on kirjattu WAL-tiedostoon. Levylle kirjoittaminen tapahtuu taustalla pienissä erissä, koska I/O-operaatiot ovat raskaita ja aikaa vieviä. Jaetussa muistissa säilytetään myös poistettujen rivien sekä muutettujen rivien vanhoja versioita, koska muut prosessit saattavat tarvita näitä arvoja. Autovacuum-prosessi poistaa vanhentuneet rivit jaetusta muistista.[31][32][33]



Kuva 6. PostgreSQL:n arkkitehtuuri.[32]

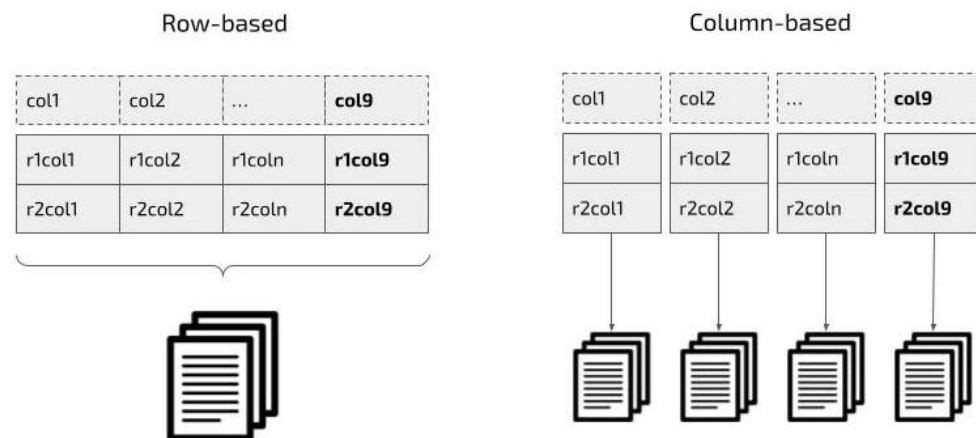
2.2.2 NoSQL

NoSQL tarjoaa mahdollisuuden tallentaa dataa muussa muodossa kuin relaatiomallin tauluina. Ensimmäiset NoSQL-tietokannat kehitettiin jo 1960-luvulla, mutta termi NoSQL syntyi vasta 2000-luvun alussa. NoSQL-tietokannoille ei ole yhtä yksiselitteistä määritelmää, vaan ne jaetaan neljään pääkategoriaan: asiakirjatietokannat, avain-arvo-tietokannat, sarakepohjaiset tietokannat ja graafitietokannat. [34]

Asiakirjatietokannat tallentavat datan dokumentteina. Yleisimmin käytetyt dokumenttiformaatit ovat JSON, BSON ja XML. Tietokannan dokumentit voivat olla sisäkkäisiä, ja tiettyjä elementtejä dokumentin sisällä voidaan indeksoida nopeamman haun mahdollistamiseksi. Asiakirjatietokannat käyttävät kokoelmia, joihin tallennetaan yleensä asiakirjoja, joilla on samankaltainen sisältö. Tämä vastaa sitä, miten SQL-tietokannat tallentavat dataa tauluihin. Asiakirjatietokannoissa on joustavat skeemat, mikä tarkoittaa, että kaikissa kokoelman asiakirjoissa ei tarvitse olla samoja kenttiä. Skeemavalidoinnin avulla kentille voidaan luoda sääntöjä, kuten sallitut tietotyypit ja arvoalueet, ja näin varmistaa, että asiakirjojen kentät noudattavat haluttua rakennetta. Asiakirjatietokannoissa on käytössä API tai kyselykieli, joiden avulla voidaan suorittaa operaatioita tietokantaan. [34][35]

Avain-arvo-tietokannat ovat tietokantatyyppejä, joissa data tallennetaan avain-arvo-muodossa, ja ne on optimoitu nopeaan tietojen lukemiseen ja kirjoittamiseen. Tietoja haetaan yksilöllisen avaimen avulla, ja arvot voivat sisältää minkä tahansa tietotyypin. Yksi yleinen käyttötapaus avain-arvo-tietokannoille on datan tallentaminen välimuistiin, joko muistiin tai levyille. [36]

Sarakepohjaiset tietokannat tallentavat datan tauluihin samalla tavalla kuin SQL-tietokannat. Ero näiden tietokantojen välillä syntyy siitä, miten data tallennetaan levyille. Sarakepohjaisissa tietokannoissa data tallennetaan sarakkeittain, kun taas SQL-tietokannoissa data tallennetaan riveittäin. Kuva 7 esittää SQL-tietokantojen ja sarakepohjaisten tietokantojen tapaa tallentaa taulujen data levyille.[36]



Kuva 7. SQL- ja sarakepohjaisten tietokantojen taulujen erot.[37]

SQL-tietokannat tallentavat yhden rivin tiedot muistiin samaan paikkaan ja ovat optimoituja rivien nopeaan lukemiseen ja kirjoittamiseen. Sarakepohjaiset tietokannat järjestävät tiedot sarakkeittain ja pitävät kaikki tiettyyn sarakkeeseen liittyvät tiedot muistissa vierekkäin. Ne ovat optimoituja sarakkeiden tehokkaaseen lukemiseen. Sarakepohjaiset tietokannat ovat erityisen hyödyllisiä analyyttisissä kyselyissä, joissa skannataan tai aggregoidaan suuria tietokokonaisuuksia, mutta tarvitaan vain muutamia sarakkeita. Koska vain tarvittavat sarakkeet luetaan tallennustilasta, I/O-operaatioihin kuluva aika on huomattavasti pienempi kuin SQL-tietokannoissa. Vaikka sarakepohjaiset tietokannat soveltuvat erinomaisesti analytiikkaan, niiden kirjoitustapa vaikeuttaa tietojen päivittämistä. Uuden rivin lisääminen tietokantaan vaatii useita kirjoitusoperaatioita, koska jokaisen sarakkeen arvo täytyy kirjoittaa levyille erilliseen paikkaan.[36]

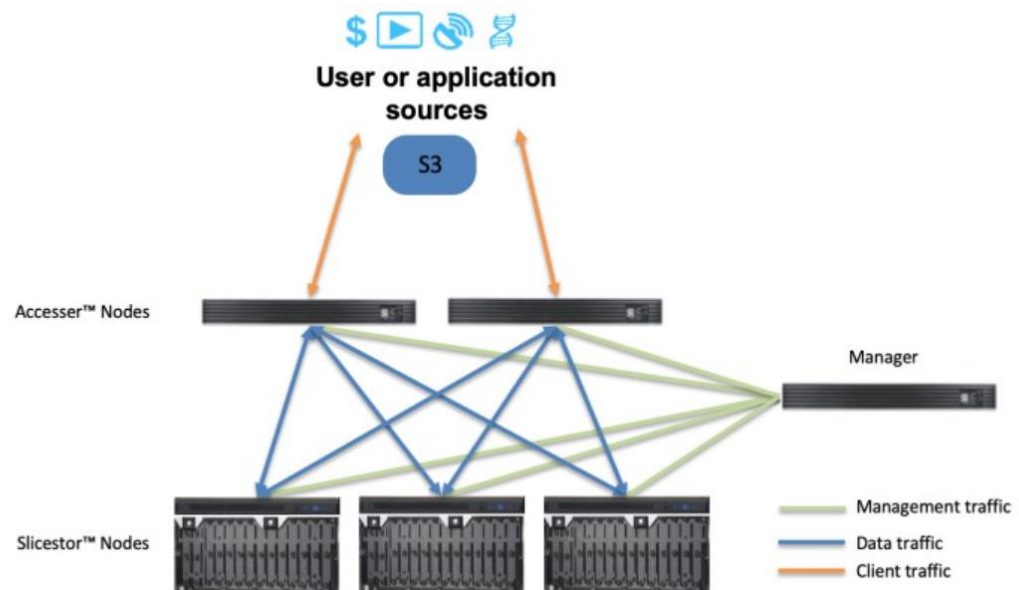
Graafitietokannat tallentavat tiedot solmuina ja reitteinä. Solmut tallentavat tyypillisesti tietoja kohteista, ja reitit tallentavat tietoja solmujen välisistä suhteista. Ne soveltuvat erityisen hyvin monimutkaisiin ja vahvasti kytkeytyneisiin tietoihin, joissa suhteet tai

niiden riippuvuudet eivät ole yksinkertaisia. Graafitietokantojen taustalla oleva tallennusmekanismi voi vaihdella. Yleisesti graafitietokannat käyttävät muita tietokantoja tallennusratkaisunaan, ja suosituimmat näistä ovat SQL-tietokannat, asiakirjatietokannat ja avain-arvo-tietokannat. [36]

2.2.3 Object storage

Object storage tarjoaa tallennusratkaisun rakenteettomalle datalle. Tiedostojen tyypillä ja koolla ei ole merkitystä, vaan kaikki tiedostot tallennetaan yhteen ämpäriin. Ämpäriissä tiedostoilla ei ole kansiorakennetta, vaan jokaisella tiedostolla on uniikki tunniste (ID). Ämpäriin pääsee käsiksi HTTP REST -rajapinnan kautta, joka mahdollistaa tiedostojen hallinnan. Jokainen Object storage -palvelua toteuttava taho voi luoda oman REST-rajapintansa, mutta käytännössä lähes kaikki noudattavat Amazon AWS S3 -ämpäriin rajapintaa. S3 oli ensimmäinen Object storage -ratkaisu pilvipalveluissa, ja sen hyvin dokumentoitu rajapinta teki siitä erittäin suosittua. Tämän seurauksena muut pilvipalvelut alkoivat tarjota samaa rajapintaa. Object storage tarjoaa erittäin hyvän skaalautuvuuden, sillä ämpäriin kokoa voidaan kasvattaa jopa exatavujen suuruiseksi. Lisäksi Object storage on kustannustehokas ratkaisu, koska laskutus perustuu ainoastaan käytetyn tallennustilan määrään. Toisin kuin perinteisissä levytallennusratkaisuissa, käyttäjää ei veloiteta ylimääräisestä varalla olevasta tallennustilasta.[38]

Nykyään jokaisella suurella pilvipalvelulla on oma versio Object storagesta, ja niiden sisäiset toteutukset voivat vaihdella. Seuraavaksi on esitelty IBM:n Object storage -toteutus. Kuva 8 esittää sen arkkitehtuurin korkealla tasolla.



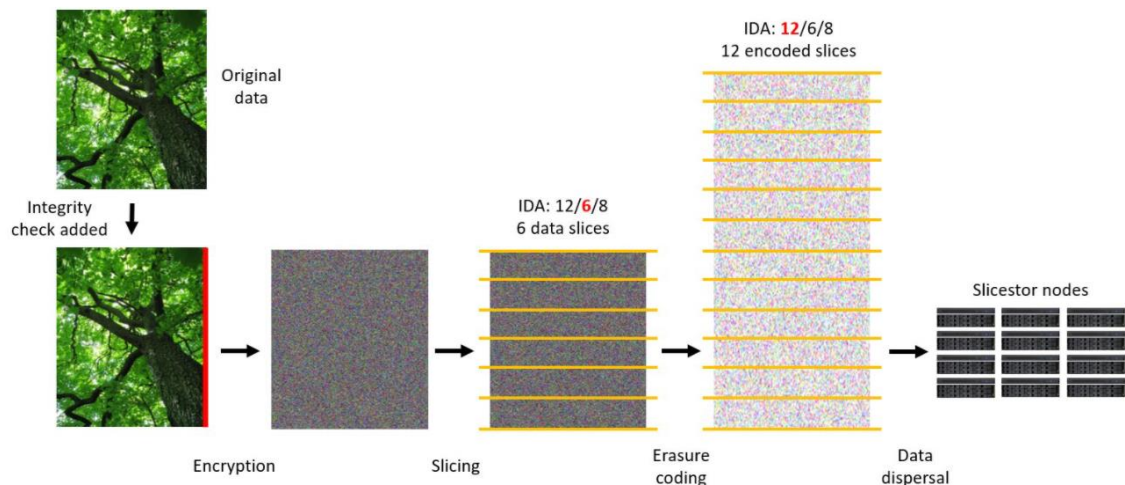
Kuva 8. IBM:n Object storagen arkkitehtuuri.[39]

Manager tarjoaa toiminnot admin-käyttäjille ja mahdollistaa järjestelmän valvonnan sekä hallinnan. Accesser tarjoaa REST-rajapinnan käyttäjille ja vastaa datan salauksesta tallennuksen yhteydessä sekä salauksen purkamisesta lukuvaiheessa. SliceStore on vastuussa tiedostosiivujen tallentamisesta, jotka se saa Accesserilta. SliceStoreiden ryhmää kutsutaan laitekokonaisuudeksi, ja laitekokonaisuuksien ryhmää puolestaan varastointialtaaksi. Ämpärit sijaitsevat sisäisessä arkkitehtuurissa varastointialtaassa. Yksi varastointiallas voi sisältää useita ämpäreitä, ja jokainen ämpäri koostuu SliceStoreista, jotka kuuluvat eri laitekokonaisuuksiin.[39]

Datan tallentamiseen käytetään Information Dispersal Algorithm (IDA) -algoritmia. Algoritmin perusidea on seuraava: tallennettava tiedosto pilkotaan paloiksi, ja palojen lukumäärää kutsutaan algoritmin leveydeksi. Kaikkia näitä paloja ei kuitenkaan tarvita tiedoston lukemiseen tai kirjoittamiseen kokonaisuudessaan, vaan pienempi määrä riittää. Lukukynnysarvo määrittää, kuinka monta palaa tarvitaan alkuperäisen datan lukemiseksi, ja kirjoituskynnysarvo määrittää, kuinka monta palaa on kirjoitettava, jotta kirjoitusoperaatio voidaan merkitä suoritetuksi. Kirjoituskynnysarvo on aina suurempi kuin lukukynnysarvo.[39]

Kuva 9 esittää tiedoston kirjoitusprosessin. Prosessi alkaa Accesser-palvelusta, joka pilkkoo yli 4 MiB:n kokoiset tiedostot neljäksi erilliseksi siivuksi. Siivut vietään yksitellen loppuprosessin läpi, kun taas pienet tiedostot vietään kokonaisina. Seuraavaksi dataan lisätään tiivistefunktion avulla tarkistusarvo eheyden varmistamiseksi, ja tarvittaessa data salataan lohkosalauksella. Data viipaloidaan ensin lukukynnysarvon

määrittämän määrän mukaisesti viipaleiksi, ja näistä viipaleista luodaan algoritmin leveyden verran paloja. Kirjoitusoperaation onnistumiseksi tulee algoritmin kirjoituskynnysarvon mukainen määrä paloja kirjoittaa, jotta operaatio voidaan merkitä suoritetuksi. Nämä palat lähetetään Slicestor-palvelimelle tallennettavaksi. Kun kirjoitusoperaatio on onnistuneesti suoritettu, puuttuvat palaset kirjoitetaan loppuun taustaprosessina. Lukuprosessi toimii käänteisesti kirjoitusprosessiin nähden. Slicestor-palvelimelta luetaan lukukynnysarvon mukainen määrä siivuja, ja näistä siivuista rekonstruoidaan alkuperäinen tiedosto.[39]



Kuva 9. Tiedoston kirjoitusprosessi IDA-algoritmissa.[39]

2.2.4 Block storage

Block storage tallentaa datan vakiokokoisina paloina, jotka se tallentaa yksilöllisen tunnusteen avulla. Tavallinen tietokoneen kiintolevy toimii käyttöjärjestelmässä Block storagen kanssa identtisesti tiedostoja tallentaessa. Block storage toteutetaan pilvipalveluissa tallennusalueverkkojen (SAN) avulla, jotka ovat yleisimpiä varastointiverkkoarkkitehtuureja. SAN yhdistää useita tallennuslaitteita yhdeksi verkoksi, johon tallennustilaa käyttävät laitteet myös liittyvät.[40]

SAN mahdollistaa verkon luomisen useilla eri protokollilla, mutta kaikki nämä protokollat käyttävät Small Computer System Interface (SCSI) -protokollaa. SCSI on vanha, vuonna 1986 standardoitu protokolla, joka määrittää tiedonsiirtoväylän sekä sen, miten siihen kytketyt laitteet kommunikoivat keskenään. Monet tietokonearkkitehtuurit sisältävät erillisen I/O-väylän, jonka avulla suoritin (CPU) kommunikoi I/O-laitteiden kanssa. Tämä väylä toteuttaa SCSI-protokollan. SCSI-väylällä voidaan yhdistää useita tallennuslaitteita ja palvelimia, mutta sen skaalautuvuus on erittäin rajallinen. Verkkoon voidaan liittää vain rajallinen määrä

laitteita, ja kaapelien maksimipituus on lyhyt. Versiosta riippuen verkkoon voidaan liittää 8–16 laitetta, ja laitteiden välinen etäisyys voi olla enintään noin 20 metriä. Koska SCSI-väylät eivät ole tehokkaita tallennusalueverkkojen luomiseen, ne on korvattu paremmilla ja skaalautuvammilla väylillä.[41]

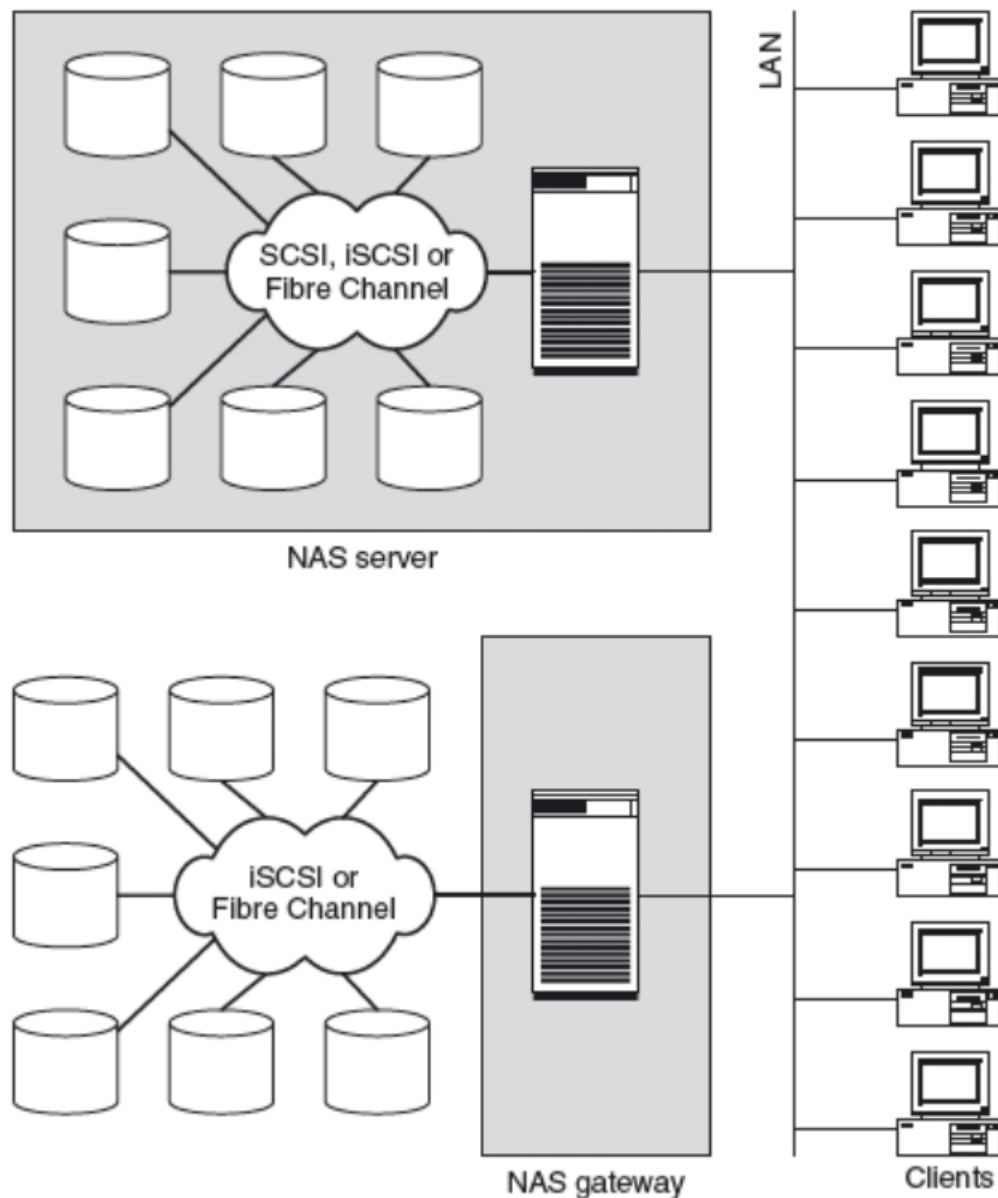
Valokuituverkot käyttävät valokuituja tiedonsiirtoon, ja Fibre Channel Protocol (FCP) on näiden verkkojen käyttämä protokolla. Valokuituverkot vastaanottavat SCSI-komentoja, muuntavat ne FCP-protokollan komennoiksi tiedonsiirron ajaksi ja muuttavat ne takaisin SCSI-komennoiksi, kun siirto on suoritettu. Tätä varten käyttöjärjestelmään on asennettava oma laiteajuri. Tallennustilaa käyttävät laitteet ja tallennuslaitteet kytketään samaan valokuituverkkoon, jolloin tiedonsiirrot eivät kuormita normaalia internetyhteyttä. Tämä protokolla on yleisimmin käytetty ja nopein ratkaisu, koska valokuituverkko on erittäin tehokas tiedostojen siirtoon. Haittapuolena on verkon korkea hinta ja sen ylläpitämiseen tarvittava erityisosaaminen. [42]

Toiseksi yleisin protokolla on iSCSI, joka käyttää internetyhteyttä valokuituverkon sijasta. iSCSI on huomattavasti edullisempi kuin valokuituverkot, mutta myös hitaampi. iSCSI siirtää SCSI-komennot TCP-protokollan avulla verkossa. iSCSI on myös tehottomampi tapa siirtää dataa, koska TCP-paketeissa on huomattavasti enemmän ylimääräistä dataa verrattuna FCP-protokollaan. Muita TCP:tä käyttäviä SAN-protokollia ovat iFCP ja FCIP, mutta ne ovat huomattavasti vähemmän käytettyjä kuin iSCSI. [41]

Pilvipalveluiden tarjoamat Block storage -ratkaisut on lähes poikkeuksetta toteutettu iSCSI-protokollan avulla, ja ne voidaan liittää virtuaalikoneisiin tavallisina hakemistoina. Block storageen tallennettu data ei ole sidottu virtuaalikoneen elinkaareen, eli data säilyy, vaikka virtuaalikone, johon Block storage oli liitetty, poistettaisiin.

2.2.5 File storage

File storage tallentaa datan hierarkkisessa rakenteessa käyttäen tiedostoja ja kansioita. File storage mahdollistaa levyn käytön useille yhtäaikaistilille käyttäjille, ja sen toteutus perustuu samoihin teknologioihin kuin tavallinen verkkolevy. Kuva 10 esittelee Network Attached Storage (NAS) -arkkitehtuurin, joka on yleisin tapa toteuttaa File storage. NAS käyttää sisäisesti samoja teknologioita kuin SAN, mutta lisää oman protokollakerroksen hallitsemaan tiedostoja. NAS-palvelin avaa IP-osoitteen, johon asiakkaat voivat ottaa yhteyden LAN-verkosta. Tiedostot liikkuvat samassa verkossa, missä muukin internetliikenne tapahtuu, ja tiedostojen siirto voi kuormittaa muuta internetliikennettä. [43]



Kuva 10. NAS-arkkitehtuuri.[44]

NAS edellyttää tiedonsiirtoprotokollan käyttämistä ja asiakkailta tiedonsiirtoprotokollan asiakasohjelmaa. Yleisimmät protokollat ovat Server Message Block (SMB) ja Network File System (NFS). SMB on alun perin IBM:n kehittämä protokolla, jonka Microsoft otti myöhemmin käyttöön Windows-käyttöjärjestelmissä. SMB on yleisesti käytössä Windows-ympäristöissä ja on oletuksena Windowsin tiedostojen jakoprotokolla. NFS on puolestaan kehitetty Sun Microsystemsin toimesta UNIX- ja Linux-ympäristöjä varten. NFS on yleisesti käytössä Linux- ja Unix-pohjaisissa järjestelmissä, ja sillä on tuki muille käyttöjärjestelmille, mukaan lukien Windows. SMB on monimutkaisempi protokolla kuin NFS, koska se tarjoaa laajempia toimintoja kuten tiedostojen lukituksen

ja metatietojen hallinnan. NFS on yleensä nopeampi ja kevyempi protokolla erityisesti Unix- ja Linux-järjestelmissä, koska se on yksinkertaisempi.[45]

2.3 Pilvipalvelut

Tässä kappaleessa käsitellään projektissä läytettävät julkisen pilven palvelut.

2.3.1 Microsoft Azure

Microsoft Azure on Microsoftin ylläpitämä pilvipalvelu, joka julkistettiin Microsoftin Professional Developers Conferencessa (PDC) lokakuussa 2008. Virallisesti se julkaistiin helmikuussa 2010 nimellä Windows Azure vaihtoehtona silloisille tunnetuille Amazonin ja Googlen pilvipalveluille. Azuren ensimmäinen versio tarjosi hyvin rajoitetun joukon palveluita, kuten pilvipalvelun ASP.NET-verkkosovellusten kehittämiseen ja käyttämiseen, Azure Blob -tietovaraston, Azure SQL -pilvitietokannan ja Azure Service Busin. Vuonna 2014 alustan nimi muutettiin nykyiseksi Microsoft Azureksi. Azuren kehitystarina alkoi jo vuonna 2005, kun Microsoft osti Groove Networksin, ja tämän jälkeen Microsoftin kehittäjät aloittivat pilvikäyttöjärjestelmän kehittämisen. Azurea koodattiin aluksi koodinimellä Red Dog, ja se oli tuolloin Windows NT:n laajennus, joka oli suunniteltu toimimaan pilvessä.[46]

Azure tarjoaa ohjelmistoa palveluna (SaaS), alustaa palveluna (PaaS) ja infrastruktuuria palveluna (IaaS). Se tukee monia erilaisia ohjelmointikieliä, työkaluja ja kehyksiä, mukaan lukien sekä Microsoftin omat että kolmannen osapuolen ohjelmistot ja järjestelmät.[46]

Azure, kuten muutkin pilvipalvelut, perustuu virtualisointitekologiaan. Tietojenkäsittelyssä virtualisointi tarkoittaa virtuaalisen version luomista jostakin tietokoneen osasta, kuten virtuaalisista tietokonelaitteistoista, tallennuslaitteista ja tietokoneverkkoresursseista. Suurin osa tietokonelaitteistoista voidaan virtualisoida ohjelmiston avulla. Tietokonelaitteisto koostuu yksinkertaisesti joukosta ohjeita, jotka on pysyvästi tai puolipysyvästi koodattu piihin. Virtualisointikerrokset toimivat siltana ohjelmiston ja laitteiston ohjeiden välillä. Ne mahdollistavat sen, että virtualisoitu laitteisto voidaan suorittaa ohjelmistossa samalla tavalla kuin fyysinen laitteisto.[47]

Pohjimmiltaan pilvipalvelut koostuvat joukosta fyysisiä palvelimia yhdessä tai useammassa palvelinkeskuksessa. Nämä palvelinkeskuksat tarjoavat virtualisoituja laitteistoja asiakkaille. Jokaisessa palvelinkeskuksessa on kokoelma palvelimia, jotka on järjestetty palvelintelineisiin. Jokainen palvelinteline sisältää useita palvelinkortteja

sekä verkkokytkimen, jotka tarjoavat verkkoyhteyden. Lisäksi telineissä on virranjakeluyksikkö (PDU), joka huolehtii sähkönjakelusta. Palvelintelineet voidaan myös ryhmitellä suuremmiksi yksiköiksi, joita kutsutaan klustereiksi. Nämä telineet tai klusterit suorittavat virtualisoituja laitteistoja käyttäjille.[47]

Joissakin palvelimissa ajetaan kuitenkin pilvihallintaohjelmistoa, joka tunnetaan kuituohjaimena. Kuituohjain on hajautettu sovellus, jolla on useita tehtäviä, kuten palveluiden allokointi, palvelimien ja niiden suorittamien palveluiden kunnon valvonta sekä palvelimien korjaaminen, jos ne kaatuvat. Kuituohjaimet ovat yhteydessä pilviorkesteriohjelmistoon, joka huolehtii palvelimista, rajapinnoista ja sisäisistä tietokannoista, joita Azure tarvitsee toimiakseen.[47]

Pilviorkesteriohjelmisto käsittelee muun muassa käyttäjien hallintakäyttöliittymän kautta tekemiä pyyntöjä, joilla varataan Azure-resursseja ja -palveluita. Käyttöliittymä tarkistaa ensin, onko käyttäjällä oikeudet pyydettyjen resurssien varaamiseen. Jos oikeudet ovat kunnossa, käyttöliittymä tarkistaa tietokannasta palvelintelineen, jolla on riittävä kapasiteetti. Sen jälkeen käyttöliittymä kehottaa kuituohjainta varaamaan resurssin käyttäjälle.[47]

Azure-alue on maantieteellinen alue, jossa sijaitsee yksi tai useampi fyysinen Azure-palvelinkeskus. Nämä palvelinkeskukset ovat osa latenssimääriteltä verkkoa, jonka tarkoituksena on tarjota käyttäjille paras mahdollinen suorituskyky ja suojaus. Monilla Azure-alueilla on käytettävyyssvyöhykkeitä, jotka ovat erillisiä palvelinkeskusten ryhmiä Azure-alueella. Yhdessä käytettävyyssvyöhykkeessä voi olla yksi tai useampi palvelinkeskus. Käytettävyyssvyöhykkeet ovat riittävän lähellä toisiaan, jotta niiden väliset yhteydet ovat matalalatenssisia, mutta riittävän kaukana toisistaan vähentääkseen riskiä, että paikalliset käyttökatkot tai sääolosuhteet vaikuttaisivat useampaan kuin yhteen vyöhykkeeseen. Käytettävyyssvyöhykkeet on yhdistetty tehokkaalla verkolla, jonka edestakainen latenssi on alle 2 ms. Lisäksi niillä on itsenäinen teho-, jäähdytys- ja verkkoinfrastruktuuri.[48]

Azure tallentaa aina useita kopioita tiedostoista suojatakseen niitä sekä suunnitelluilta että suunnittelemattomilta tapahtumilta, kuten laitteistohäiriöiltä tai sähkökatkoilta. Tätä tiedostojen suojaamista kopioimalla kutsutaan redundanssiksi. Azure tarjoaa neljä erilaista redundanssiasetusta, joiden avulla voidaan hallita, missä tiedostojen kopiot säilytetään.

Azuren tallennusratkaisuja hallitaan tallennustilin kautta. Tallennustilille voidaan luoda useita Azuren tarjoamia tallennuspalveluita. Tallennustilin redundanssiasetus koskee kaikkia tilille luotuja tallennuspalveluita. Tallennustilin tiedostot replikoidaan aina kolme

kertaa Azure-alueella, johon tallennustili on luotu. Azure tarjoaa kaksi vaihtoehtoa tiedostojen replikointiin tällä Azure-alueella:[49]

Paikallisesti redundantti tallennus (LRS) kopioi tiedostot synkronisesti kolme kertaa yhden palvelinkeskuksen sisällä. LRS on edullisin redundanssivaihtoehto, mutta se tarjoaa heikoimman suojan tiedostojen tuhoutumiselle verrattuna muihin vaihtoehtoihin. LRS suojaa tiedostoja palvelinteline- ja asemavikoja vastaan. Jos palvelinkeskuksessa kuitenkin tapahtuu katastrofi, kuten tulipalo tai tulva, kaikki LRS:ää käyttävän tallennustilin kopiot saattavat kadota.[49]

Alueellinen redundantti tallennus (ZRS) kopioi tiedostot synkronisesti kolmen käytettävyyssvyöhykkeen välillä käytössä olevan Azure-alueen sisällä. ZRS mahdollistaa luku- ja kirjoitustoiminnot, vaikka kaikki käytettävyyssvyöhykkeet, joihin tiedosto on kopioitu, eivät olisi käytettävissä. Kirjoituspyyntö tallennustilille, joka käyttää ZRS:ää, suoritetaan synkronisesti. Kirjoitustoiminto katsotaan onnistuneeksi vasta sen jälkeen, kun tiedostot on kirjoitettu kaikkiin replikoihin kolmella käytettävyyssvyöhykkeellä. ZRS tarjoaa tiedostoille paremman saatavuuden ja redundanssin käyttämällä useampia käytettävyyssvyöhykkeitä kuin LRS.[49]

Azure mahdollistaa myös tiedostojen kopioimisen asynkronisesti toiseen Azure-alueeseen. Geo-redundantti tallennus (GRS) toimii samalla tavalla kuin LRS, mutta tallentaa toiset kolme kopiota asynkronisesti toiseen Azure-alueeseen. Geo-alue-redundantti tallennus (GZRS) puolestaan toimii samalla tavalla kuin ZRS, mutta kopioi tiedostot toiseen Azure-alueeseen samalla tavalla kuin GRS.[49]

2.4 Tietoturva

Tässä kappaleessa käsitellään tietoturvaan liittyviä protokollia.

2.4.1 RSA

Rivest–Shamir–Adleman (RSA) on julkisen avaimen salausjärjestelmä ja yksi vanhimmista yleisesti käytetyistä salausmenetelmistä. RSA salaa tai allekirjoittaa tiedot kahdella eri avaimella: julkisella ja yksityisellä avaimella. Julkinen avain on kaikkien käytettävissä, kun taas yksityinen avain pidetään salassa. Julkisella avaimella salattujen tietojen salaus voidaan purkaa vain yksityisellä avaimella.[50]

Allekirjoittamisessa käytetään yksityistä avainta allekirjoituksen luomiseen, ja kuka tahansa voi vahvistaa allekirjoituksen aitouden julkisen avaimen avulla. Kahden avaimen käytön vuoksi julkisen avaimen salaus tunnetaan myös epäsymmetrisenä

salauksena, koska salaus ja salauksen purku tapahtuvat eri avaimilla. Toisin kuin epäsymmetrisessä salauksessa, symmetrisessä salauksessa samaa avainta käytetään sekä salauksen luomiseen että purkamiseen.[50][51]

Nimi RSA tulee Ron Rivestin, Adi Shamirin ja Leonard Adlemanin sukunimistä, jotka julkaisivat algoritmin vuonna 1977. RSA:n turvallisuus perustuu siihen, kuinka vaikeaa on käytännössä jakaa kahden suuren alkuluvun tulo tekijöihinsä. RSA-salauksen murtaminen tunnetaan RSA-ongelmana. Tätä ongelmaa ei ole onnistuttu ratkaisemaan, jos käytössä on riittävän suuri avain. RSA on kuitenkin suhteellisen hidas algoritmi, minkä vuoksi sitä ei yleensä käytetä suoraan tietojen salaamiseen, vaan sen avulla salataan jaettuja avaimia, joita käytetään symmetrisessä salauksessa.[50][51]

RSA:n peruseriaate on, että on suhteellisen helppoa löytää kolme erittäin suurta positiivista kokonaislukua e , d ja n , niin että kaikille kokonaisluvuille m , jotka täyttävät ehdon $0 < m < n$, pätee seuraava yhtälö:

$$(m^e)^d \equiv m \pmod{n}$$

Kun annetaan vain e ja n , on erittäin vaikea löytää d . Kaavassa Kokonaisluvut n ja e muodostavat julkisen avaimen, d on yksityinen avain ja m on viesti. Koska e ja d voidaan vaihtaa kaavassa päikseen, mahdollistaa se viestien allekirjoittamisen ja allekirjoituksen tarkistamisen samaa algoritmia käyttäen. Julkinen avain n muodostetaan kahden satunnaisen suuren alkuluvun tulona, ja sen pituus määrittää julkisen avaimen pituuden. e on positiivinen satunnainen kokonaisluku, jonka arvon valintaväli määritetään Carmichaelin funktion avulla. Yksityinen avain d saadaan myös laskettua Carmichael funktion ja julkisen avaimen avulla. Viesti m ($0 < m < n$) saadaan salattua laskemalla seuraava salakirjoitus c käyttäen julkista avainta e ja n kaavalla.

$$c \equiv m^e \pmod{n}$$

Salattu viesti c voidaan myöhemmin purkaa käyttämällä yksityistä avainta d , jolloin alkuperäinen viesti m saadaan takaisin kaavalla: [52]

$$m \equiv c^d \pmod{n}$$

2.4.2 TLS

TLS on julkisen avaimen salausmenetelmä, joka on luotu vanhentuneen SSL-protokollan päivitettyä versiona. SSL:n ensimmäistä versiota ei koskaan julkaistu

tietoturvaongelmien takia, mutta ensimmäinen julkaistu versio, SSL 2.0, julkaistiin vuonna 1995. SSL 2.0:ssa havaittiin pian tietoturvaongelmia, jotka korjattiin seuraavana vuonna julkaistussa SSL 3.0 -versiossa. TLS 1.0 julkaistiin vuonna 1999, eikä siinä ollut merkittäviä eroja SSL 3.0 -versioon verrattuna. Vuonna 2006 julkaistu TLS 1.1 toi lisää tietoturvaparannuksia. TLS 1.2 julkaistiin vuonna 2008, ja nykyinen uusin versio, TLS 1.3, julkaistiin vuonna 2018. Uudemmat versiot ovat parantaneet tietoturvaa poistamalla protokollasta vanhentuneita ja murtuneita salausalgoritmeja sekä pakottamalla käyttämään vahvempia avaimia salausalgoritmeissa.[53]

TLS käyttää X.509-sertifikaattistandardin varmenteita viestintäkumppanin todentamiseen. Kun asiakas ottaa yhteyden palvelimeen, palvelin lähettää varmenteen asiakkaalle. Tärkeitä tietoja varmenteessa ovat verkkotunnuksen nimi, jolle varmenne on myönnetty, varmenneviranomainen (CA), joka on myöntänyt varmenteen ja allekirjoittanut sen, varmenteen voimassaoloaika sekä palvelimen julkinen avain. [54][55]

CA on kolmannen osapuolen palvelu, joka tallentaa, allekirjoittaa ja myöntää digitaalisia varmenteita. Molemmat osapuolet, sekä asiakas että palvelin, luottavat CA:han. Asiakkaalla on luotettujen varmenneviranomaisten lista ja niiden julkiset avaimet. Asiakas pystyy vahvistamaan palvelimen varmenteen oikeellisuuden tarkistamalla CA:n tekemän allekirjoituksen, joka on luotu CA:n yksityisellä avaimella. [54][55]

Ennen varmenteen allekirjoittamista varmenneviranomaisen on varmistettava, että palvelin todella omistaa verkkotunnuksen, jolle varmenne on pyydetty. Tämä todentaminen voidaan suorittaa esimerkiksi DNS-tietueiden avulla. CA:n tekemän allekirjoituksen avulla asiakas voi luottaa palvelimen identiteettiin. Varmenteen voi myös allekirjoittaa omalla yksityisellä avaimella, jolloin puhutaan itseallekirjoitetuista varmenteista. Tällöin asiakas ei voi varmistaa palvelimen aitoutta varmenneviranomaisten kautta, vaan luottamus täytyy todentaa toisella tavalla. [54][55]

TLS-kättely on prosessi, joka käynnistää TLS:ää käyttävän viestintäistunnon. TLS-kättelyn aikana kaksi kommunikoivaa osapuolta vaihtavat viestejä vahvistaakseen toisensa, sopiakseen käytettävistä salausalgoritmeista ja sopiakseen istuntoavaimista. TLS-versioissa ennen versiota 1.3 prosessi etenee seuraavasti:

Ensimmäiseksi asiakas lähettää viestin palvelimelle, jossa se ilmoittaa tukemansa TLS-versiot ja salausalgoritmit. Lisäksi asiakas lähettää satunnaisesti generoidun numeron. Palvelin vastaa valitsemalla korkeimman TLS-version, jota molemmat osapuolet tukevat, sekä yhteisesti tukeman salauksen. Palvelin lähettää myös oman

satunnaisesti generoidun numeron sekä SSL-sertifikaatin asiakkaalle, jonka avulla asiakas voi varmistaa sertifikaatin aitouden.[56]

Kun sertifikaatin aitous on varmistettu, asiakkaalla on useita tapoja jakaa symmetriseen salaukseen käytettävä avain palvelimen kanssa. Yksinkertaisin tapa on RSA-kättely, jossa asiakas lähettää palvelimelle avaimen salattuna palvelimen julkisella avaimella. Ainoastaan palvelin pystyy purkamaan tämän salauksen yksityisellä avaimellaan.[56]

Monimutkaisempi tapa avaimen vaihtoon on käyttää Diffie–Hellman-avaintenvaihtoprotokollaa (DH). DH-protokollassa avainta ei lähetetä salattuna toiselle osapuolelle, vaan molemmat osapuolet laskevat yhteisen salaisuuden julkisia arvoja käyttäen. DH-protokollassa valitaan alkuluku p ja sen generaattori q , jotka jaetaan julkisesti osapuolten välillä. Molemmat osapuolet valitsevat itselleen salaisen luvun x ja laskevat siitä julkisen luvun X kaavalla:

$$X \equiv q^x \pmod{p}$$

Osapuolet jakavat julkiset luvut toisilleen ja laskevat yhteisen salaisuuden s kaavalla:

$$s \equiv X^x \pmod{p}$$

jossa X on toisen osapuolen julkaisema julkinen luku ja x on oma salainen luku. [58]

TLS 1.3 yksinkertaistaa kättelyprosessia merkittävästi ja parantaa tietoturvaa poistamalla protokollasta epäturvallisia algoritmeja. Yhteisen salaisuuden jakamisessa TLS 1.3 vaatii eteenpäin turvattavuutta. Tämä tarkoittaa, että vaikka palvelimen yksityinen avain paljastuisi tulevaisuudessa, sillä ei voida purkaa aiempia salauksia. Tämän vuoksi aiemmin mainittu RSA-kättely ei ole enää mahdollista, ja Diffie-Hellman-protokollassa (DH) vaaditaan, että jokaiselle yhteydelle käytetään aina uusia julkisia parametrejä. [57]

Asiakas lähettää TLS 1.3:ssa ensimmäisessä viestissään satunnaisesti generoidun numeron, tukemansa symmetriset salausalgoritmit sekä valitsee menetelmän yhteisen salaisuuden jakamiseen ja laskee siihen tarvittavat julkiset arvot. Tässä vaiheessa asiakas ei voi tietää varmasti, tukeeko palvelin valittua menetelmää, mutta koska vaihtoehtoja on vähemmän kuin aikaisemmissa versioissa, on todennäköistä, että palvelin tukee asiakkaan valintaa. [56]

Palvelin vastaa tähän viestiin valitulla symmetrisellä salausalgoritmilla, omalla satunnaisluvullaan sekä menetelmällä yhteisen salaisuuden jakamiseen, johon se on laskenut omat julkiset arvonsa. Palvelin pystyy tässä vaiheessa laskemaan yhteisen salaisuuden ja on valmis aloittamaan symmetrisen salauksen käytön. Palvelimen viesti toimii samalla kuittauksena aloittaa salattu yhteys. [56]

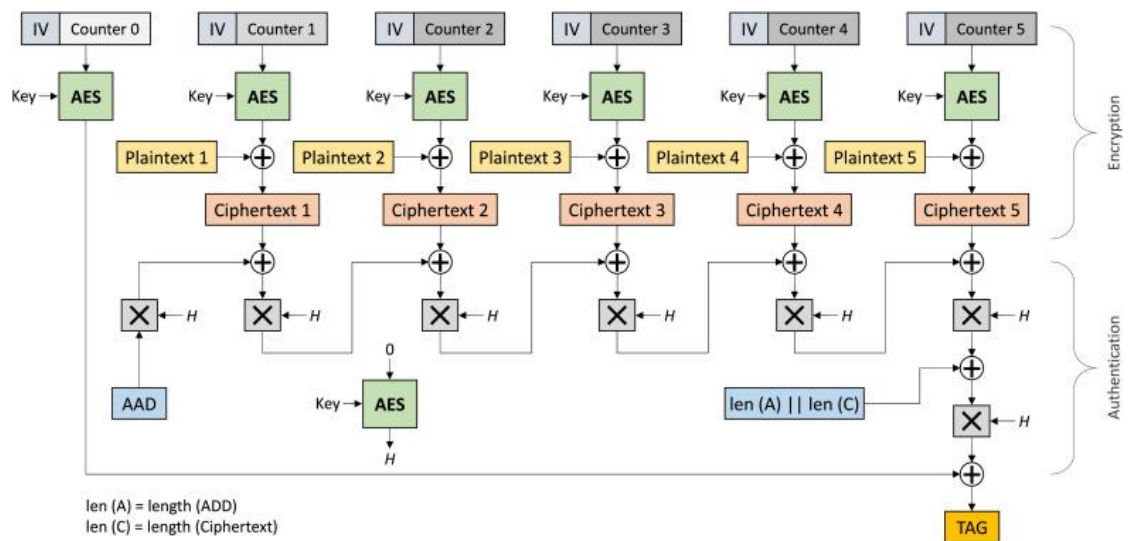
Asiakas laskee jaetun salaisuuden palvelimen vastauksen perusteella ja on valmis käyttämään symmetristä salausta. Asiakas varmistaa vielä palvelimen varmenteen aitouden ja lähettää kuittausviestin palvelimelle. Kättely on nyt suoritettu, ja yhteys jatkuu symmetrisellä salauksella. [56]

TLS 1.3 tukee myös nopeampaa TLS-kättelyn versiota, joka ei vaadi lainkaan edestakaisia viestejä. Jos asiakas ja palvelin ovat aiemmin muodostaneet yhteyden, ne voivat luoda toisen jaetun salaisuuden ensimmäisen yhteyden aikana käytettäväksi seuraavaa yhteyttä varten. Palvelin lähettää asiakkaalle myös istuntolipun yhteyden aikana. Seuraavan yhteyden alkaessa asiakas voi käyttää istuntolippua ja yhteyden jatkamista varten luotua salaisuutta, jolloin TLS-salaus voidaan pystyttää ilman perinteistä kättelyvaihetta.[56]

Symmetrisessä salauksessa käytetään joko lohko- tai virtasalausalgoritmia. Lohkosalausalgoritmi ottaa vakiokokoisien lohkon dataa ja salaa sen salatekstiksi. Virtasalauksessa jokainen salattavan tekstin bitti salataan yksitellen salausnumerovirran vastaavalla bitillä, jotta saadaan salatekstivirran bitti. Lohkosalausta voidaan käyttää virtasalauksen tapaan, jos hyödynnetään alustusvektoreita. Alustusvektori salataan lohkosalauksella, ja salattu alustusvektori yhdistetään salattavaan tekstiin. TLS-protokollan symmetrisessä salauksessa käytetään lohkosalausta virtasalauksen kaltaisesti. Salauksen lisäksi algoritmin täytyy pystyä todentamaan salatun datan aitous eli varmistaa, ettei kukaan ole muokannut dataa siirron aikana. Tällaisia salausjärjestelmiä, jotka pystyvät sekä salaamaan datan että todentamaan sen aitouden, kutsutaan todennetuksi salaukseksi (AE). Yleisin AE-algoritmi on Galois/Counter Mode (GCM), jota TLS-protokolla käyttää. TLS 1.3 mahdollistaa myös toisen AE-algoritmin, Counter with CBC-MAC (CCM), jota aiemmat TLS-versiot eivät tukeneet.[59][60][61]

TLS 1.2 ja sitä aikaisemmat versiot käyttävät useita eri lohko- ja virtasalausalgoritmeja, mutta TLS 1.3 versiossa käytetään vain yhtä lohkosalausta (AES) ja yhtä virtasalausta (ChaCha20). AES (Advanced Encryption Standard), joka tunnetaan myös alkuperäisellä nimellään Rijndael, on Yhdysvaltain kansallisen standardisointi- ja teknologiainstituutin (NIST) vuonna 2008 vahvistama spesifikaatio lohkosalaukseen. AES on nykyään yksi käytetyimmistä lohkosalausalgoritmeista, ja siitä on olemassa variaatiot 128-, 192- ja 256-bittisille avaimille.[62][56]

Kuva 11 esittelee lohko- ja virtasalausalgoritmeja, jotka TLS 1.2 ja TLS 1.3 käyttävät.



Kuva 11. AES-GCM-lohkokaavio.[63]

Lohkokaavion yläosassa lasketaan salattu data AES-algoritmilla, ja alaosassa lasketaan varmenne salatusta datasta, jonka avulla voidaan todentaa, että dataa ei ole muokattu siirron aikana. Salauksessa käytetään alustusvektoria (IV), johon lisätään lohkonumero, ja tämä summa salataan AES-algoritmin avulla. Varmenne lasketaan Galois'n kuntien avulla, ja siihen luodaan avain AES-salauksessa käytetyn avaimen pohjalta. Varmenteen laskemiseen käytetään salattua dataa, alustusvektoria sekä mahdollista ylimääräistä dataa. AES-GCM mahdollistaa myös ylimääräisen salaamattoman datan lähettämisen salatun datan kanssa ja pystyy todentamaan tämän aitouden ottamalla sen mukaan varmenteen laskentaan. Tämä on merkitty kaavioon nimellä AAD (Additional Authenticated Data). [47][50]

Kuva 12 esittää F5 Labsin tekemän The 2021 TLS Telemetry Report -tutkimuksen tuloksia TLS:n käytetyimmistä salausalgoritmeista. Tutkimuksessa on mukana miljoona käytetyintä nettisivustoa. Salausalgoritmit on esitetty muodossa yhteisen salaisuuden jakaminen – symmetrinen salaus – MAC. Tutkimuksen mukaan neljä yleisintä algoritmia käyttävät kaikki symmetriseen salaukseen AES-GCM-algoritmia, joka esiteltiin edellisessä kappaleessa. Kun neljän suosituimman algoritmin käyttöprosentit lasketaan yhteen, havaitaan, että AES-GCM suojaa 93,2 % kaikista TLS-yhteyksistä. Lisäksi kuvasta käy ilmi, että yhteisen salaisuuden jakamiseen käytetään DH-protokollan monimutkaisempaa elliptisiin käyriin perustuvaa versiota, joka on merkitty kuvaan lyhenteellä ECDHE (Elliptic Curve Diffie-Hellman Ephemeral).[64]

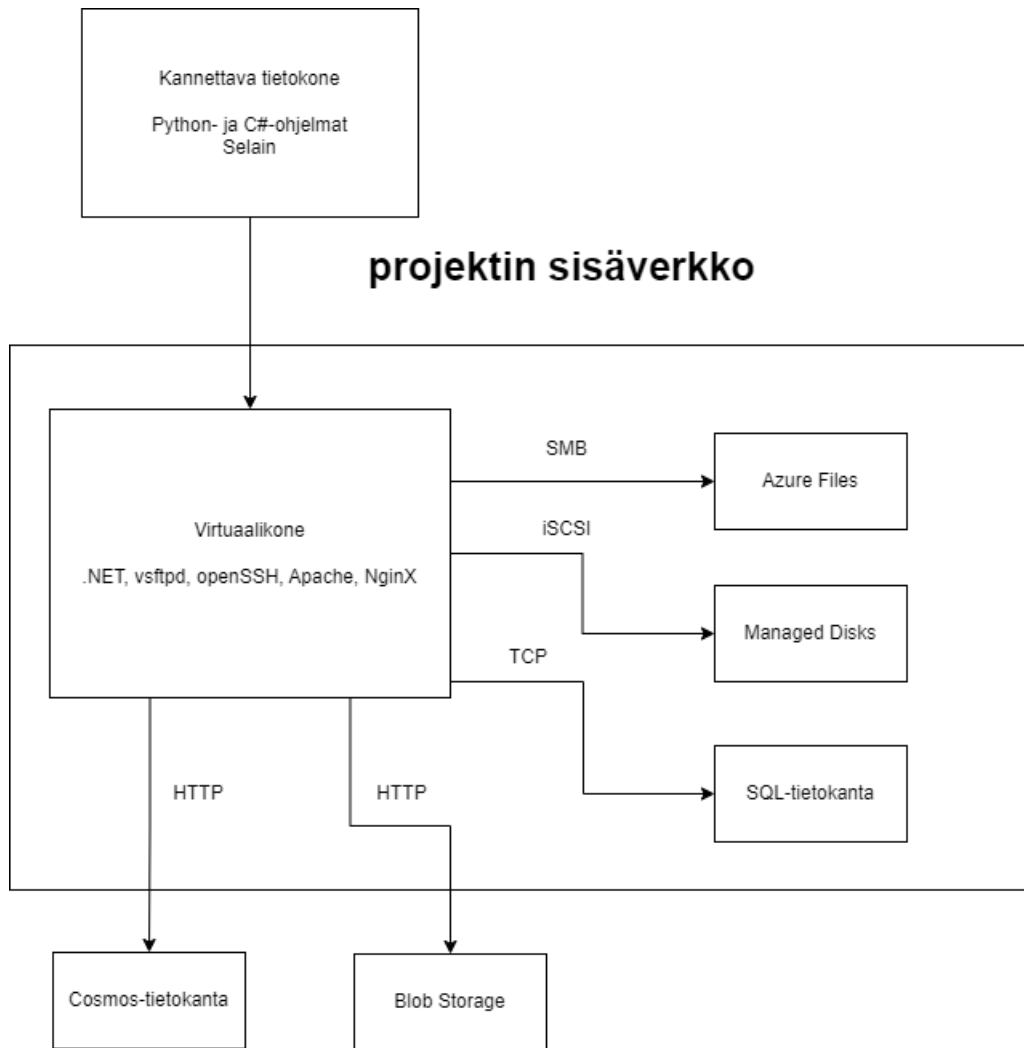
TABLE 1: THE MOST POPULAR SELECTED CIPHER SUITES IN THE TOP MILLION SITES

Protocol	Cipher suite chosen by web server	Proportion of top 1M sites
TLS 1.3	TLS_AES_256_GCM_SHA384	56.8%
TLS 1.2	ECDHE-RSA-AES256-GCM-SHA384	18.4%
TLS 1.2	ECDHE-RSA-AES128-GCM-SHA256	12.6%
TLS 1.3	TLS_AES_128_GCM_SHA256	5.4%
TLS 1.2	ECDHE-RSA-AES256-SHA384	1.9%
TLS 1.2	ECDHE-RSA-CHACHA20-POLY1305	1.4%
TLS 1.3	TLS_CHACHA20_POLY1305_SHA256	0.5%
TLS 1.2	ECDHE-ECDSA-AES256-GCM-SHA384	0.4%
TLS 1.2	ECDHE-ECDSA-CHACHA20-POLY1305	0.4%
TLS 1.2	DHE-RSA-AES256-GCM-SHA384	0.4%
TLS 1.2	ECDHE-ECDSA-AES128-GCM-SHA256	0.3%
TLS 1.0	DHE-RSA-AES256-SHA	0.3%

Kuva 12. Suosituimmat TLS-salausalgoritmit vuonna 2021.[64]

3. TESTAUSYMPÄRISTÖ

Työssä esitellään useita pienempiä testausprojekteja, jotka voidaan jakaa karkeasti kahteen osioon: tiedostojen siirtoon ja tiedostojen talletukseen. Kuva 13 esittelee kokonaisjärjestelmän arkkitehtuurin, joka kattaa molemmat osiot. Järjestelmä on luotu Azure-pilvipalvelua hyödyntäen, ja testejä suoritetaan Azuren ulkopuolella kannettavalla tietokoneella. Azuressa on käytössä virtuaalikone, jossa pyörii testaukseen vaadittavia palvelimia. Virtuaalikoneen käyttöjärjestelmänä on Ubuntu 22.04 ja virtuaalikoneessa on 1 CPU ja 8 GB RAM-muistia. Käyttöliittymä on toteutettu Angular-ohjelmointikehyksellä ja se noudattaa Single Page Application (SPA) -arkkitehtuuria. Nginx-palvelinta käytetään SPA-käyttöliittymän jakamiseen internetiin. Virtuaalikoneelle on asennettu myös vsftpd-palvelin, joka on FTP-palvelin ja mahdollistaa myös FTPS-yhteyden. WebDAV-palvelin on toteutettu Apache-palvelimen avulla. OpenSSH-palvelinta käytetään SSH-yhteyden muodostamiseen, jonka kautta luodaan SFTP-yhteys. Virtuaalikoneeseen on liitetty Block storage ja File storage, jotka on yhdistetty osaksi virtuaalikoneen tiedostojärjestelmää. Azuren tarjoama Block storage on nimeltään Managed Disks ja File storage on nimeltään Azure Files. Object storageen ja Cosmos-tietokantaan pääsee käsiksi julkisen HTTP-rajapinnan kautta. Azuren Object storage on nimeltään Blob Storage, ja Cosmos-tietokanta on Azuren toteuttama NoSQL-tietokanta, jossa data on tallennettu JSON-dokumentteina. Virtuaalikoneen kanssa samassa virtuaalisessa sisäverkossa on myös PostgreSQL-palvelin. Virtuaalikone kommunikoi SQL-palvelimen kanssa C#-ohjelmointikielellä toteutetun asiakasohjelman avulla.



Kuva 13. Projektin kokonaisarkkitehtuuri.

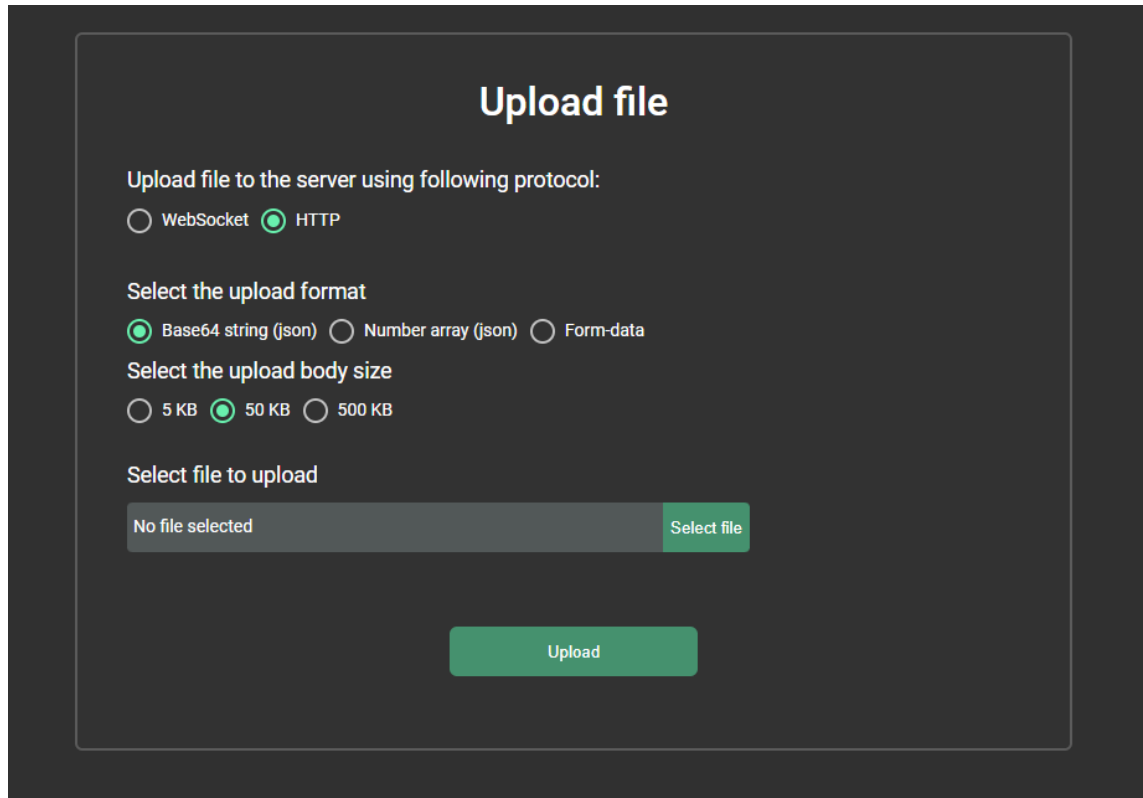
3.1 Tiedostojen siirto

Tiedostojen siirtoon liittyvät testit on jaettu useampaan pienempään ryhmään eri tiedostonsiirtotapojen perusteella. Tiedostojen siirtämiseen on käytetty selainta sekä Python- ja C#-koodia. FTP-, FTPS- ja SFTP-siirrot on toteutettu kahdella eri ohjelmointikielen kirjastolla: Pythonilla ja C#:lla.

Python-ohjelmointikielellä FTP- ja FTPS-kirjastot ovat osa standardikirjastoa, joten erillisiä kolmannen osapuolen kirjastoja ei tarvita. SFTP-yhteys on toteutettu Paramiko-kirjastolla, joka on kolmannen osapuolen avoimen lähdekoodin kirjasto. C#-koodissa yhteyksien luomiseen on käytetty kolmannen osapuolen NuGet-paketteja. FluentFTP-kirjastoa on hyödynnetty FTP- ja FTPS-yhteyksien luomiseen, ja SSH.NET-kirjastoa SFTP-yhteyteen. Projektissa on ladattu erikokoisia tiedostoja kummallakin ohjelmointikielellä. Tiedostojen koot ovat olleet kilotavujen ja 10 megatavun välillä. Saman kokoinen tiedosto on ladattu viisi kertaa, ja jokaisen latauksen latausaika on

kirjattu. Näistä latausajoista on laskettu keskiarvo kullekin tiedostokoolle, ja keskiarvoja on käytetty tulosten analysoimiseen.

Selaimella toimivia protokollia, eli HTTP- ja WebSocket-protokollia, on testattu selainpohjaisen käyttöliittymän sekä C#-ohjelmointikielen avulla. Kuva 14 on näyttökuva käyttöliittymän HTTP-latauksesta.



Kuva 14. Projektin HTTP-latausnäköymä.

Käyttöliittymässä voi valita erilaisia asetuksia käytettäviin protokolliin. HTTP-latauksessa tiedostoa ei lueta kokonaan selaimen muistiin, vaan se luetaan palasina. Jos tiedostoa ei paloittelaisi, koko tiedosto pitäisi lukea selaimen RAM-muistiin, mikä rajoittaisi merkittävästi lähetettävän tiedoston maksimikokoa. Tiedoston paloittelun ansiosta lähetettävän tiedoston kokoa ei rajoiteta. Käyttöliittymässä on valittavissa 5 kB:n, 50 kB:n ja 500 kB:n kokoiset palaset. Jokainen palanen lähetetään omana POST-pyyntönään palvelimelle, ja viimeisen palasen jälkeen palvelin kokoaa palaset yhdeksi kokonaiseksi tiedostoksi.

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 kahdeksanbittisinä numeroina taulukkotietorakenteessa. Data voidaan siirtää eri koodauksia käyttäen. Base64-koodauksessa binääridata muunnetaan merkkijonoksi, ja Base64-koodisto käyttää 64-

merkkistä aakkostoa, johon binääridata muunnetaan. Number array -valinta lähettää binäärisen datan merkkijonona ilman koodausta.

WebSocket-protokollassa voidaan valita siirrettävän datan formaatti: merkkijono tai binääri. Merkkijono käyttää UTF-merkistöä siirrettävän datan koodauksena, kun taas binääri käyttää binääriformaattia tiedonsiirtoon. WebSocket-protokollaa käytetään myös C#-koodissa .NET:n omalla WebSocket-kirjastolla sekä SignalR-kirjastolla, joka on osa .NET:ä.

Käyttöliittymää ohjataan testeissä Selenium-kirjastolla, joka mahdollistaa selaimen käytön automatisoinnin. Seleniumin avulla selainta ohjataan toistamaan aina samat valinnat. Kun tiedosto on ladattu onnistuneesti selaimella, latausaika näkyy käyttöliittymässä, ja Selenium poimii sen talteen.

HTTP- ja WebDAV-protokollia testataan myös C#-koodilla. HTTP-testaukseen käytetään .NET:n omaa kirjastoa, ja WebDAV:n testaukseen käytetään WebDav.Client-NuGet-pakettia.

3.2 Tiedostojen tallennus

Tiedostojen tallennusta varten on toteutettu .NET-palvelin ja C#-komentoriviohjelmistoja, jotka suorittavat tallennusoperaatiot. Tallennusoperaatiot suoritetaan useaan kertaan, ja niille lasketaan keskiarvot sekä joissain testeissä myös keskihajonnat. Managed Disks -tallennusratkaisut on liitetty osaksi virtuaalikoneen hakemistoa SCSI-protokollan avulla, kun taas Azure Files on liitetty SMB-protokollalla. Muihin tallennusratkaisuihin päästään käsiksi TCP- tai HTTP-protokollan kautta, ja niihin otetaan yhteys C#-ohjelmointikielen kirjastoilla. Testeissä tutkitaan useiden eri suorituskyykyisten Managed Disks- ja Azure Files -tallennusratkaisujen versioiden suorituskyykyä sekä redundanssiasetuksen vaikutusta suorituskyykyyn. Tästä syystä useita tallennusratkaisuja on liitetty samaan virtuaalikoneeseen.

4. PROJEKTIN TULOKSET

Projektin tulokset on jaettu suorituskyvyn, kustannusten ja tietoturvan mukaan. Kustannusvertailussa tarkastellaan tallennusratkaisujen kustannuksia Azure-pilvipalvelussa sekä vertaillaan eri tallennusratkaisujen hintoja. Lisäksi tiedonsiirron kustannuksia arvioidaan epäsuorasti käytetyn internetkaistan perusteella. Tehokkuusvertailussa keskitytään tallennus- ja lataustapojen suorituskäyttöön ajallisesta näkökulmasta. Tietoturva-vertailussa tarkastellaan käytettyjen ratkaisujen tietoturvaa ja vertaillaan tutkimuksia, jotka käsittelevät pilvipalveluita käyttävien sovellusten tietoturvaa.

4.1 Kustannusvertailu

Kustannusvertailu on jaettu kahteen osaan: pilvipalveluiden tallennusratkaisuihin ja tiedostojen siirtoon. Tallennusratkaisujen kustannusvertailu tehtiin vertailemalla niiden hintoja erikokoisilla tallennustiloilla sekä analysoimalla, mistä komponenteista tallennusratkaisun hinta muodostuu. Tiedostojen siirron kustannuksia mitataan epäsuorasti arvioimalla, kuinka paljon siirrettävän tiedoston koko kasvaa eri protokollien käytön aikana. Siirrettävän datan määrä voi vaikuttaa internetyhteyden hintaan, ja lisäksi pilvipalvelut laskuttavat virtuaalikoneille saapuvasta ja lähtevästä datasta. Tiedostojen siirron tutkimus tehtiin käyttäen WireShark-ohjelmistoa, joka on avoimen lähdekoodin pakettianalysaattori. Wireshark on monialustainen työkalu, joka käyttää Packet Capturea (PCAP) pakettien sieppaamiseen.

4.1.1 Tallennusratkaisuiden kustannukset

Tässä luvussa esitellään Azuren tarjoamien tallennusratkaisujen hinnat sekä niiden komponentit, joista kokonaiskustannus muodostuu. Hinnat on laskettu East US -Azure-alueella, ja ne käyttävät paikallisesti redundanttia tallennusta (LRS), ellei toisin mainita. Hinnat ovat olleet voimassa 9.9.2024.

Cosmos-tietokanta on NoSQL-tietokanta ja vektoritietokanta, joka tallentaa dokumentteja JSON-muodossa. Lisäksi Cosmos-tietokannalla on tuki PostgreSQL-tietokannalle Citus-laajennuksen avulla, mutta sitä ei käsitellä tässä työssä. JSON-dokumentit indeksoidaan, ja niitä voi hakea ja muokata kyselyillä. Azure Cosmos

laskuttaa kolmesta eri käyttötyypistä: laskentakapasiteetista, tallennustilasta ja kaistanleveydestä.[65][66]

Azure tarjoaa ilmaiseksi 25 GB tallennustilaa jokaiselle Azure-tilille. Laskennan laskutus perustuu pyyntöyksikköihin (RU), jotka mittaavat tietokantaoperaatioiden vaatimien laskenta-, muisti- ja I/O-resurssien kulutusta. Eri tietokantaoperaatiot kuluttavat pyyntöyksikköjä eri määrän. Vaadittavan RU-määrän määrittävät useat tekijät, kuten kirjoitettavan tai luettavan tiedon koko. Indeksien käyttö lisää tarvittavien pyyntöyksikköjen määrää. Vaadittavien pyyntöyksikköjen määrä on deterministinen, eli sama operaatio samalla tietokannalla vaatii aina saman määrän pyyntöyksikköjä. [65][66]

Azure Cosmos laskuttaa pyyntöyksikköinä sekunnissa (RU/s) tai pelkästään käytetyistä pyyntöyksiköistä, riippuen valitusta skaalautuvuusmallista:

- **Palveliton malli** laskuttaa vain käytetyistä pyyntöyksiköistä, 0,235 € miljoonalta pyyntöyksiköltä.
- **Kiinteän suorituskyvyn malli** kiinnittää tietokannalle kiinteän RU/s-arvon, ja siitä veloitetaan 0,0075 € tunnilta sataa RU/s-yksikköä kohden.
- **Automaattisen skaalauksen malli** skaalaa RU/s-arvon automaattisesti kuorman mukaan 10 %:n ja 100 %:n välillä annetusta maksimi-RU/s-arvosta. Tunnin maksimi RU/s-arvosta veloitetaan 0,008 € tunnilta sataa RU/s-yksikköä kohden.

Tallennustilan hinta on 0,235 € kuukaudessa gigatavulta. Kaistanleveydestä veloitetaan vain tietokannasta ulos siirrettävästä datasta. Ensimmäiset 5 GB ovat ilmaisia, minkä jälkeen hinta on 0,05 € gigatavulta. [65][66]

Azure tarjoaa täysin ylläpidetyn SQL-tietokannan ja varmistaa sen saatavuuden ja suorituskyvyn. Tietokannoissa laskutetaan sekä laskentakapasiteetista että tallennustilasta. Laskentaan on tarjolla kaksi eri mallia: virtuaaliyttimeen (vCore) ja tietokantatapahtumayksikköön (DTU) perustuva ostomalli. [67][68][69]

Virtuaaliyttimeen (vCore) perustuva malli tarjoaa vaihtoehdot varattujen tai palvelimettomien laskentamallien välillä. Varatussa laskentamallissa voidaan valita tarkka määrä virtuaaliytimiä, jotka ovat aina varattuna laskentaa varten. Palvelimettomassa laskentamallissa virtuaaliytimien määrä skaalautuu automaattisesti määritettävän laskenta-alueen mukaan. Palvelimeton laskentamalli sulkee tietokannan automaattisesti epäaktiivisina aikoina, jolloin tietokannassa ei ole avoimia istuntoja, ja tällöin laskutusta laskentakapasiteetista ei tapahdu. Tietokanta käynnistyy uudelleen,

kun uusi istunto luodaan. Virtuaaliytimiä on tarjolla eri arkkitehtuureilla, ja niiden hinnat vaihtelevat hieman. Virtuaaliytimeksi voidaan valita 2–128 ydintä, ja yhden ytimen hinta on noin 100–120 euroa kuukaudessa. Palvelimeton malli laskuttaa ytimen käytöstä sekuntihintaan, kun taas varattu malli käyttää minuuttihintaa. [67][68][69]

Tietokantatapahtumayksikkö (DTU) edustaa suorittimen, muistin, lukemisten ja kirjoitusten abstraktia mittaa. DTU-pohjaisessa ostomallissa valitaan tietty laskentateho, joka ilmoitetaan käytettävissä olevina tietokantatapahtumayksikköinä. Laskentatehoon on kiinnitetty kiinteä määrä tallennustilaa, varmuuskopioiden säilytysaika sekä kiinteä hinta. Lisämuistia voi ostaa perusmuistin lisäksi. Laskentatehoa voi kasvattaa lisäämällä DTU-yksiköjä tietokannan luomisen jälkeen, mikä aiheuttaa vain lyhyen käyttökatkon. DTU-pohjainen laskentateho voidaan muuntaa virtuaaliytimeksi suhteella 100 DTU-yksikköä vastaa yhtä virtuaaliydintä. Laskentateho on valittavissa 5–400 DTU-yksikön väliltä, ja tallennustila vaihtelee 2 GB:n ja 4 TB:n välillä. Hinnat vaihtelevat 4,5 € ja 14 515 € välillä kuukaudessa. [67][68][69]

Virtuaaliytimeen perustuvassa mallissa tallennustila maksaa 0,232 euroa kuukaudessa gigatavulta. DTU-pohjaisessa mallissa tietty määrä levytilaa sisältyy hintaan, ja lisälevytila maksaa 0,16 € kuukaudessa gigatavulta. [67][68][69]

Azure Blob Storage on Microsoftin tarjoama Object storage -ratkaisu, joka on suunniteltu rakenteettoman datan tallentamiseen. Azure Blob Storage tarjoaa HTTP-rajapinnan, jonka kautta tiedostoja voidaan hallita ja muokata. Useille ohjelmointikielille on saatavilla valmiita asiakaskirjastoja sekä CLI-työkaluja. Blob Storage laskuttaa sekä tallennustilasta että operaatioista, ja hinnat määräytyvät valitun palvelumallin mukaan. [70][71]

Azure Blob Storagessa on viisi palvelumallia: korkealuokkainen, kuuma, lämmin, kylmä ja arkisto. Tallennustilan gigatavuhinnat nousevat arkistomallista kohti korkealuokkaista, kun taas operaatioiden hinnat halpenevat vastaavasti. Korkealuokkainen malli tarjoaa alhaisimman latenssin, ja latenssi kasvaa, kun siirrytään kohti arkistomallia. [70][71]

Korkealuokkaisen mallin tallennustila maksaa 0,13869 € gigatavulta, mikä on lähes kymmenen kertaa kalliimpaa kuin kuuman mallin tallennustila. Hinnat laskevat edelleen kohti arkistoa, jossa tallennustilan hinta on vain 0,00092 € gigatavulta. Korkealuokkainen malli tarjoaa halvimmat luku- ja kirjoitusoperaatiot. Kirjoitusoperaatiot maksavat 0,0211 € kymmentätuhatta operaatiota kohden, mikä on noin kuusi kertaa halvempi kuin arkistomallin hinta. Lukuoperaatioiden hinnat vaihtelevat huomattavasti

mallien välillä. Korkealuokkaisessa mallissa lukuoperaatiot maksavat 0,0017 € kymmentätuhatta operaatiota kohden, kun taas kylmän mallin hinta on noin 100-kertainen ja arkistomallin hinta noin 4000-kertainen verrattuna korkealuokkaiseen malliin. [70][71]

Azure Files on Microsoftin tarjoama File storage -palvelu, joka mahdollistaa tiedostojen yhtäaikaista käyttöä eri käyttöjärjestelmien ja virtuaalikoneiden välillä. Palvelumalleja on neljä: korkealuokkainen, tapahtumaoptimoitu, kuuma ja kylmä. Korkealuokkainen malli tarjoaa alhaisimman latenssin, ja latenssi kasvaa siirryttäessä kohti kylmää mallia. Korkealuokkaisen mallin tallennustila on kalleinta, ja hinta laskee siirryttäessä kohti kylmää mallia. Korkealuokkaisen mallin hinta on 0,1480 € varatulta gigatavulta, ja kylmän mallin hinta on noin kymmenen kertaa halvempi. Luku- ja kirjoitusoperaatiot ovat korkealuokkaisessa mallissa ilmaisia, mutta muissa malleissa maksullisia. Tapahtumaoptimoidussa mallissa on halvimmat luku- ja kirjoitusoperaatiot, ja hinnat nousevat siirryttäessä kohti kylmää mallia. Kylmän mallin luku- ja kirjoitusoperaatioiden hinnat ovat keskimäärin noin kymmenen kertaa kalliimmat kuin tapahtumaoptimoidussa mallissa. [72]

Azure Managed Disks on Microsoftin tarjoama Block storage -ratkaisu, jota voidaan käyttää virtuaalikoneiden kanssa. Levyvaihtoehtoja on neljä: Ultra Disk, korkealuokkainen SSD, SSD ja HDD. Ultra Disk tarjoaa pienimmän latenssin ja on uuden sukupolven SSD-levy. Korkealuokkaisen SSD:n ja tavallisen SSD:n välillä on pieni ero suorituskyvyssä ja hinnassa. [73]

Levyille on ilmoitettu, kuinka monta kirjoitus-/lukuoperaatiota voidaan suorittaa sekunnissa (IOPS) sekä tiedonsiirtonopeus, joka ilmaistaan megatavuina sekunnissa (MB/s). Azure tarjoaa mahdollisuuden parantaa hetkellisesti levyn IOPS- ja MB/s-suorituskykyä ns. purskeominaisuuden avulla. Purske voi kestää enintään 30 minuuttia yhtämittaisesti, ja sen käyttö kuluttaa krediittejä. Kun kaikki krediitit on käytetty, purske loppuu. Kun levyä käytetään alle sen IOPS- ja MB/s-suorituskyvyn, krediittejä kertyy uutta pursketta varten. [73]

Pienillä, alle 128 GB levyillä, korkealuokkaisen SSD:n suorituskyky on huonompi kuin tavallisen SSD:n, mutta korkealuokkaisen SSD:n purskeominaisuus on huomattavasti tehokkaampi. Suuremmissa levyissä korkealuokkainen SSD on tehokkaampi kuin vastaavan kokoinen tavallinen SSD-levy. Korkealuokkaisen SSD:n hinta on noin kaksi kertaa suurempi kuin vastaavan kokoisen tavallisen SSD-levyn. Korkealuokkainen SSD laskuttaa lisäksi IO-operaatioista 0,001850 € jokaista 10 000 operaatiota kohden

tunnissa, mutta IO-operaatioiden hinnalla on kuukausittainen maksimi, joka vastaa suunnilleen levyn tallennuskapasiteetin hintaa. [73]

HDD-levyn hinta on noin puolet vastaavan kokoisen SSD-levyn hinnasta. HDD:n IOPS-suorituskyky on lähes sama kuin vastaavan kokoisen SSD:n, mutta datansiirtonopeus on huomattavasti alhaisempi. Ultra Disk -levyn laskutus eroaa hieman muista levyistä: käyttäjä voi valita vapaasti haluamansa IOPS- ja MB/s-suorituskyvyn sekä levyn tallennuskoon. Jokaisesta valitusta komponentista laskutetaan erikseen, ja levyn kokonaishinta muodostuu näiden summasta. [73]

Eri tallennusratkaisujen hintoja vertaillaan kolmella eri tallennuskoon perusteella: 16 GB, 256 GB ja 1 TB. Tallennusratkaisujen pelkästä tallennustilasta laskuttama kuukausihinta on esitetty taulukossa 1. Taulukkoon on valittu 3 palvelumallia jokaisesta tallennusratkaisusta.

Taulukko 1. Tallennusratkaisuiden laskuttama kuukausihinta LSR-redundanssiasetuksella.

	16 GB	256 GB	1 TB
Managed Disks/ korkealuokkainen SSD	2,22 €	31,96 €	113,61 €
Managed Disks/ SSD	1,11 €	17,76 €	71,01 €
Managed Disks/ HDD	0,70 €	10,48 €	37,87 €
Files/ korkealuokkainen	2,37 €	37,89 €	148 €
Files/ tapahtuma optimoitu	0,88 €	14,2 €	55,5 €
Files/ viileä	0,22 €	3,55 €	13,9 €
Blob Storage/ korkealuokkainen	2,22 €	35,50 €	138,69 €
Blob Storage/ kuuma	0,27 €	4,28 €	16,7 €
Blob Storage/ arkisto	0,01 €	0,24 €	0,92 €
Cosmos	3,71 €	59,40 €	232 €
SQL	3,71 €	59,40 €	232 €

Taulukosta 1 käy ilmi, että tietokantojen (Cosmos ja SQL) hinnat ovat samat ja selvästi korkeammat kuin muiden tallennusratkaisujen. Kaikkien muiden tallennusratkaisujen kalleimpien vaihtoehtojen hinnat ovat suhteellisen lähellä toisiaan. Halvempien tallennusratkaisujen hinnat kuitenkin eroavat toisistaan enemmän. Kaikista halvin

ratkaisu on Blob Storagen arkistomalli, jonka hinta on useita kymmeniä kertoja halvempi kuin esimerkiksi halvimman Managed Disksin tai Files-palvelun. Blob Storagen arkiston IO-operaatiot ovat kuitenkin kymmeniä kertoja kalliimmat kuin muiden tallennusratkaisujen, joten se sopii parhaiten tiedostoille, joita ei tarvitse usein muokata tai lukea.

Tallennusratkaisujen IO-kuormasta syntyvien kustannusten vertailu on haastavampaa, koska eri tallennusratkaisut käyttävät erilaisia mittayksiköitä operaatioiden vaadittavan suorituskyvyn mittaamiseen. Taulukossa 2 on esitetty eri tallennusratkaisujen hinta miljoonalle lukuoperaatiolle ja miljoonalle kirjoitusoperaatiolle. Joillekin tallennusratkaisuille on jouduttu arvioimaan hinta, koska tallennusratkaisu ei suoraan ilmoita yksittäisten operaatioiden hintaa. Arvioidun hinnan edessä on käytetty tilde-merkkiä (~).

Taulukko 2. Tallennusratkaisuiden laskutus IO-operaatioille LSR-redundanssiasetuksella.

	Miljoona lukuoperaatiota	Miljoona kirjoitusoperaatiota
Managed Disks/ korkealuokkainen SSD	0 €	0 €
Managed Disks/ SSD	0,19 €	0,19 €
Managed Disks/ HDD	0 €	0 €
Files/ korkealuokkainen	0 €	0 €
Files/ tapahtuma optimoitu	0,14 €	1,39 €
Files/ viileä	1,21 €	12,02 €
Blob Storage/ korkealuokkainen	0,17 €	2,11 €
Blob Storage/ Kuuma	0,47 €	6,01 €
Blob Storage/ arkisto	600,0 €	12,02 €
Cosmos	0,23 €	~0,50 €
SQL	~0,34 €	~0,70 €

Cosmos-tietokannan lukuoperaatioiden hinnan voi laskea suoraan, koska jokainen lukuoperaatio kuluttaa aina yhden RU-resurssin. Kirjoitusoperaatio puolestaan voi käyttää eri määrän RU-resursseja riippuen tietokannan koosta ja kirjoitettavan tiedon

määrästä. Taulukossa on arvioitu, että kirjoitusoperaatio käyttäisi keskimäärin 2 RU-resurssia.[65]

SQL-tietokannan IO-operaatioiden hinta on vieläkin vaikeampi arvioida, koska käytössä oleva abstrakti laskentamitta, DTU-yksikkö, laskutetaan eri tavalla. DTU-yksiköistä veloitetaan käytettävien yksiköiden määrän perusteella kuukaudessa. Taulukossa hinta on arvioitu 5 DTU-yksikön tietokannalle, olettaen, että yksi lukuoperaatio käyttää yhden yksikön ja kestää yhden sekunnin. Näin 5 DTU-yksikön tietokanta pystyy suorittamaan noin 13 miljoonaa lukuoperaatiota kuukaudessa. Kuukauden hinta on jaettu 13:lla, jolloin saadaan miljoonan lukuoperaation hinta. Kirjoitusoperaatioiden on arvioitu kuluttavan resursseja tuplasti, ja näin ollen on päädytty taulukossa esitettyyn hintaan. Nämä hinnat edustavat minimihintoja ja edellyttävät, että tietokantaa käytetään 100 % kapasiteetilla. Todelliset hinnat ovat todennäköisesti korkeampia, riippuen tietokannan käyttöasteesta.

Kun vertailee taulukkoa 1 ja taulukkoa 2, huomaa selkeästi logiikan kylmien ja kuumien tallennusratkaisujen välillä. Kuumat ratkaisut tarjoavat tehokkaan IO-suorituskyvyn ja kalliimman tallennustilan, mutta IO-operaatiot ovat halvempia. Kylmät ratkaisut puolestaan ovat erittäin edullisia tallennustilan osalta, mutta IO-operaatiot ovat kalliita, ja niiden latenssi on korkea.

Tallennusratkaisun kustannusten optimointiin vaikuttaa useita tekijöitä. On tärkeää pohtia, kuinka paljon IO-operaatioita suoritetaan ja ovatko pieni latenssi sekä hyvä suorituskyky kriittisiä ominaisuuksia. Jos IO-operaatioita on paljon, mutta suorituskyvyn tarve ei ole kriittinen, optimaalisin hintaratkaisu löytyy lämpimän päädyn ratkaisusta. Jos IO-operaatioita on vähän ja nopeus ei ole tärkeä tekijä, kannattaa valita kylmä malli.

Taulukoista erottuu erityisesti Blob Storagen arkistomalli. Sen tallennustilakustannukset ovat merkittävästi halvempia kuin muiden ratkaisujen, mutta lukuoperaatioiden kustannukset ovat noin tuhat kertaa kalliimpia kuin minkään muun ratkaisun. Tietokantojen (Cosmos ja SQL) kustannukset ovat korkeammat kuumien ratkaisujen kohdalla, mutta ne mahdollistavat monimutkaisten kyselyjen suorittamisen ja turvallisen rinnakkaiskäytön.

Redundanssiasetuksen vaikutus hintoihin on esitetty taulukoissa 3 ja 4. Niissä on samat tallennusratkaisut ja tallennuskoot kuin taulukossa 1, ja taulukot sisältävät pelkästään tallennustilasta laskutettavat kuukausihinnat. Taulukossa 3 redundanssiasetus on ZRS, ja taulukossa 4 GRS. Jos tallennusratkaisua ei ole saatavilla kyseisellä redundanssiasetuksella, hinnan kohdalla on viiva. ZRS tallentaa

tiedot kolmeen eri käytettävyyssyöhykkeeseen, kun taas LRS käyttää ainoastaan yhtä palvelinkeskusta. GRS puolestaan tallentaa kolme kopiota yhdelle käytettävyyssyöhykkeelle ja toiset kolme kopiota asynkronisesti taustalla toiseen Azure-alueeseen.

Taulukko 3. Tallennusratkaisuiden laskuttama kuukausihinta ZRS-redundanssiasetuksella.

	16 GB	256 GB	1 TB
Managed Disks/ korkealuokkainen SSD	3,24 €	51,23 €	182,15 €
Managed Disks/ SSD	1,62 €	25,88 €	103,51 €
Managed Disks/ HDD	-	-	-
Files/ korkealuokkainen	2,88 €	46,00 €	179,70 €
Files/ tapahtuma optimoitu	1,08 €	17,25 €	67,4 €
Files/ viileä	0,27 €	4,33 €	16,90 €
Blob Storage/ korkealuokkainen	2,88 €	46,00 €	179,70 €
Blob Storage/ kuuma	0,37 €	5,99 €	23,4 €
Blob Storage/ arkisto	-	-	-
Cosmos	3,71 €	59,40 €	232 €
SQL	-	-	-

Taulukosta 3 ilmenee, että Managed Diskin ja Blob Storagen halvimmat ratkaisut eivät tue tiedostojen kopioimista useammalle käytettävyyssyöhykkeelle. Tietokannoista Cosmos tukee ZRS-redundanssiasetusta, ja tallennustilan hinta on sama kuin LRS-redundanssiasetuksella. RU-resurssien määrä kuitenkin kerrotaan 1.25, kun ZRS on käytössä. SQL-kanta tukee ZRS:ää ainoastaan varmuuskopioinnissa, jolloin varmuuskopiot voidaan sijoittaa eri käytettävyyssyöhykkeille. Kun vertaillaan tallennusratkaisujen LRS- ja ZRS-hintoja, havaitaan, että suurimmat erot ovat Managed Disks -palvelussa. Hinnat kasvavat 45 % tai 60 % tallennuskokojen ja palvelumallien mukaan. Files-palvelun hinnat nousevat vähiten, jokaisessa vaihtoehdossa noin 22 %. Blob Storagen hinnat nousevat 30–40 %

Taulukossa 4 on esitetty kuukausihinnat GRS-redundanssiasetuksella. Vain viidessä tutkituista tallennusratkaisujen palvelumalleista on tuki GRS-asetukselle. Kun

vertaillaan hintoja LRS- ja GRS-redundanssiasetusten välillä, havaitaan, että Cosmos-tietokannassa hinta säilyy samana, mutta käytettyjen RU-resurssien määrä kerrotaan niiden Azure-alueiden määrällä, joihin tiedostot on kopioitu. Muiden palveluiden hinnat nousivat huomattavasti: Files/tapahtumaoptimoitu nousi vähiten, 62 %, ja Blob Storage/arkisto eniten, noin 202 %.

Taulukko 4. Tallennusratkaisuiden laskuttama kuukausihinta GRS-redundanssiasetuksella.

	16 GB	256 GB	1 TB
Managed Disks/ korkealuokkainen SSD	-	-	-
Managed Disks/ SSD	-	-	-
Managed Disks/ HDD	-	-	-
Files/ korkealuokkainen	-	-	-
Files/ tapahtuma optimoitu	1,43 €	23,01 €	89,90 €
Files/ viileä	0,43 €	6,91 €	27,00 €
Blob Storage/ korkealuokkainen	-	-	-
Blob Storage/ kuuma	0,66 €	10,60 €	41,4 €
Blob Storage/ arkisto	0,04 €	0,68 €	2,69 €
Cosmos	3,71 €	59,40 €	232 €
SQL	-	-	-

Azure tarjoaa vielä neljännen redundanssiasetuksen GZRS, jolla on kaikista heikon tallennusratkaisuiden tuki. Taulukoiduista tallennusratkaisuista pelkästään Files/tapahtuma optimoidulla, Files/ viileällä ja Blob Storage/ kuumalla on tuki sille. Kun verrataan näiden palveluiden GRS ja GZRS hintoja keskenään niin huomataan että Files/ viileä ja Blob Storage/ kuuma hinnat nousevat alle 3 % ja Files/ tapahtuma optimoidun hinta nousee noin 22 %.

4.1.2 Tiedostojen siirron kustannukset

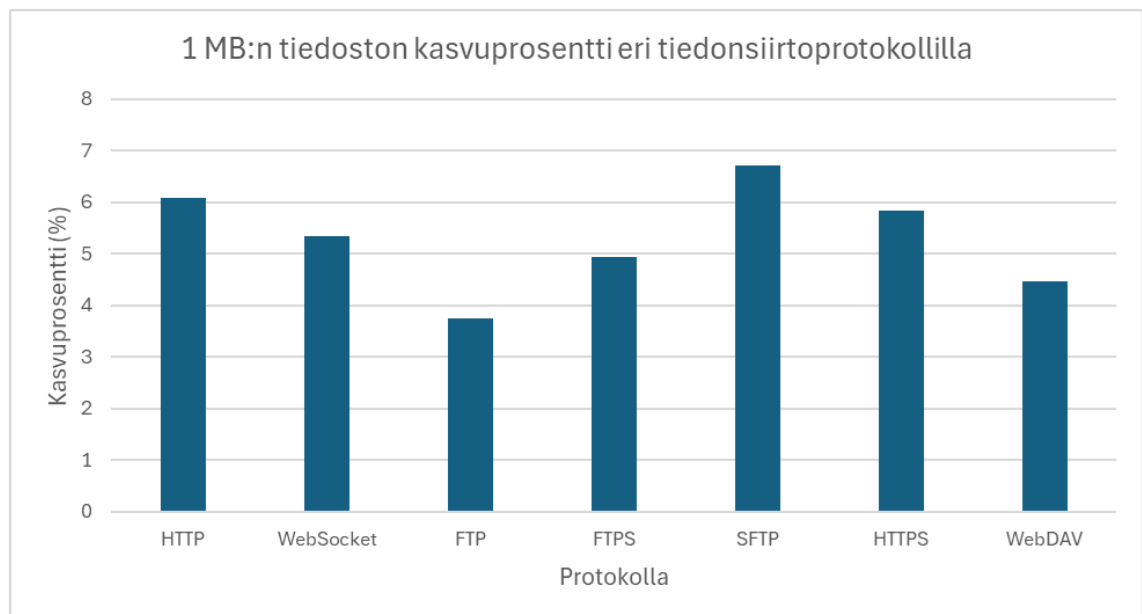
Tiedostojen siirron kustannuksia vertaillaan epäsuorasti arvioimalla, kuinka paljon eri siirtoprotokollat kasvattavat tiedoston kokoa siirron aikana. Siirrettävän datan määrän

kasvu aiheuttaa lisäkustannuksia verkkoliikenteen kasvaessa, sillä pilvipalvelut veloittavat esimerkiksi virtuaalikoneille saapuvasta ja lähtevästä datasta.

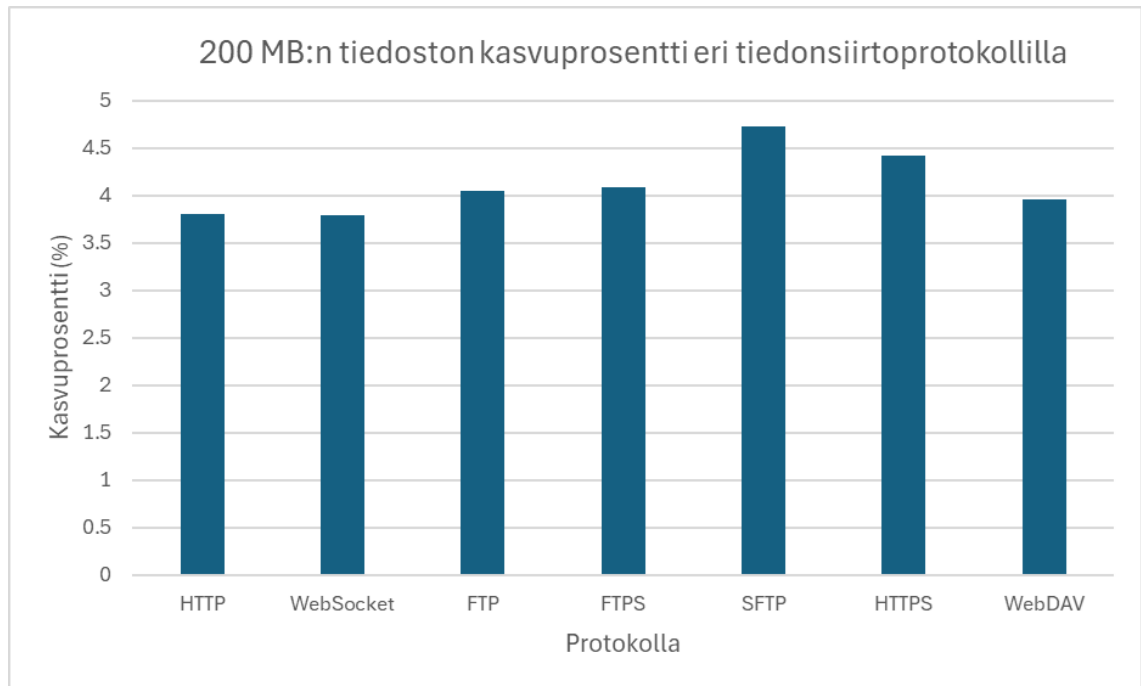
Tutkimuksessa käytettiin Wireshark-nimistä ohjelmistoa, joka on avoimen lähdekoodin pakettianalysaattori verkkoliikenteen tutkimiseen. Wireshark on monialustainen ohjelmisto, joka käyttää Packet Capture (PCAP) -protokollaa pakettien sieppaamiseen. PCAP kopioi verkossa liikkuvia datapaketteja niiden kulkiessa verkon läpi ja tallentaa nämä kopiot analysointia varten. Wireshark tarjoaa käyttöliittymän kaapattujen pakettien analysointiin, ja tämän käyttöliittymän avulla projektin tietoliikennettä on tutkittu.

Projektissa tutkittiin tietoliikennettä tiedoston siirtämisessä molempiin suuntiin: palvelimelle ja palvelimelta kulkevaa liikennettä. Molempiin suuntiin kulkenut data laskettiin yhteen, ja vertailu tehtiin tällä summatulla arvolla. Vertailuun sisältyivät kaikki projektissa käytetyt protokollat, ja niitä käytettiin C#-kirjastoilla. Vertailu suoritettiin kahdella erikokoisella tiedostolla, joiden koot olivat 1 MB ja 200 MB.

Kuva 15 esittelee vertailun 1 MB:n tiedoston kanssa, ja Kuva 16 puolestaan vertailun 200 MB:n tiedoston kanssa. Tiedostojen koon kasvu kuvissa on ilmoitettu prosentuaalisena kasvuna verrattuna alkuperäiseen tiedostoon.



Kuva 15. 1 MB:n tiedoston kasvuprosentti siirrettäessä.



Kuva 16. 200 MB:n tiedoston kasvuprosentti siirrettäessä.

1 MB:n tiedosto kasvaa hieman enemmän siirtäessä kuin suurempi 200 MB:n tiedosto kaikilla muilla protokollilla paitsi FTP:llä, jolla tiedoston koko pysyy lähes samana. 1 MB:n tiedoston kasvuprosentti vaihtelee välillä 3,7 % – 6,7 %, kun taas 200 MB:n tiedoston kasvuprosentti on 4,0 % – 4,7 %. SFTP aiheuttaa suurimman kasvun kummallakin tiedostokoolla. Pienemmällä tiedostolla FTP tuottaa pienimmän kasvun, kun taas suuremmalla tiedostolla WebSocket aiheuttaa vähiten kasvua.

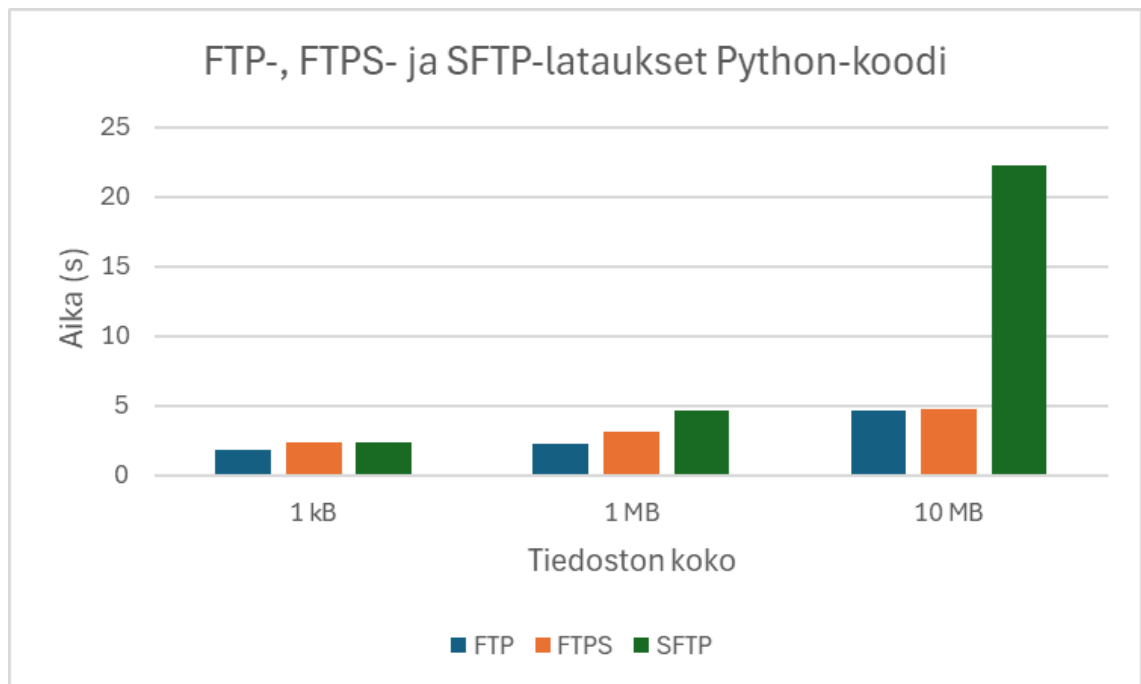
Salatut protokollat kasvattavat tiedoston kokoa eniten suuremmalla tiedostolla. Pienemmällä tiedostolla SFTP ja HTTPS kasvattavat eniten, mutta FTPS aiheuttaa vähemmän kasvua kuin salaamattomat HTTP tai WebSocket. Pienellä tiedostokoolla kasvuun vaikuttavat enemmän esimerkiksi protokollan yhteydenavauksen ja käyttäjähallinnan prosessit. Esimerkiksi HTTP-protokollassa kasvua aiheuttaa HTTP-pyyynnön vastaus, joka pienemmällä tiedostokoolla muodostaa noin 1 % tiedoston kokonaiskoosta.

4.2 Tehokkuusvertailu

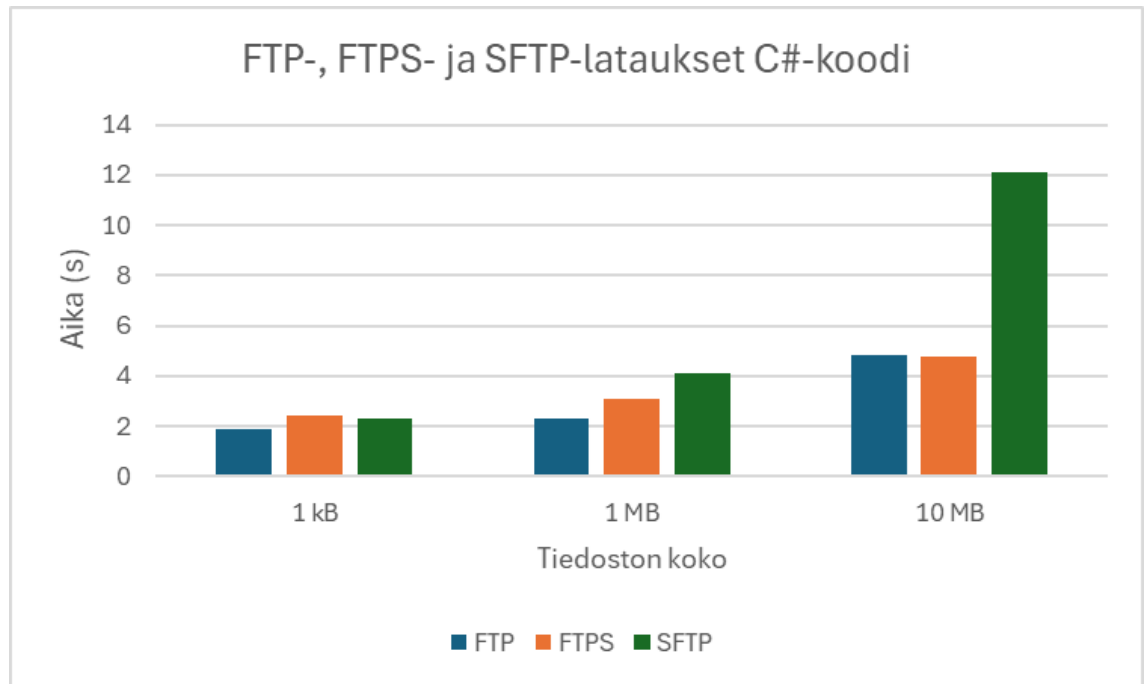
Tämä luku on jaettu kahteen osaan: tiedostojen siirtoon ja tiedostojen tallennukseen. Molemmat osat sisältävät erilaisia suorituskykytestejä.

4.2.1 Tiedostojen siirto

Kuva 17 esittää FTP-, FTPS- ja SFTP-latausten latausaikojen tulokset Python-ohjelmointikielellä toteutettuna ja Kuva 18 esittää samat lataukset C#-kielellä. Kuvista nähdään, että SFTP on selvästi FTP:tä ja FTPS:ää hitaampi molemmilla ohjelmointikielillä suurimmilla tiedostoilla. Pienellä tiedostokoolla SFTP ei ole hitaampi kuin FTP tai FTPS, mutta ero kasvaa voimakkaasti tiedoston koon kasvaessa. Python-koodissa SFTP:n suorituskyky on selvästi heikompi kuin C#-koodissa, mutta FTP- ja FTPS-latausten kohdalla ohjelmointikielellä ei ole merkittävää eroa. FTP:n ja FTPS:n suorituskyvyssä on vain pieniä eroja. Pienillä tiedostoilla FTP on hieman nopeampi, koska aloitusprosessi on lyhyempi, sillä yhteyttä ei salata.

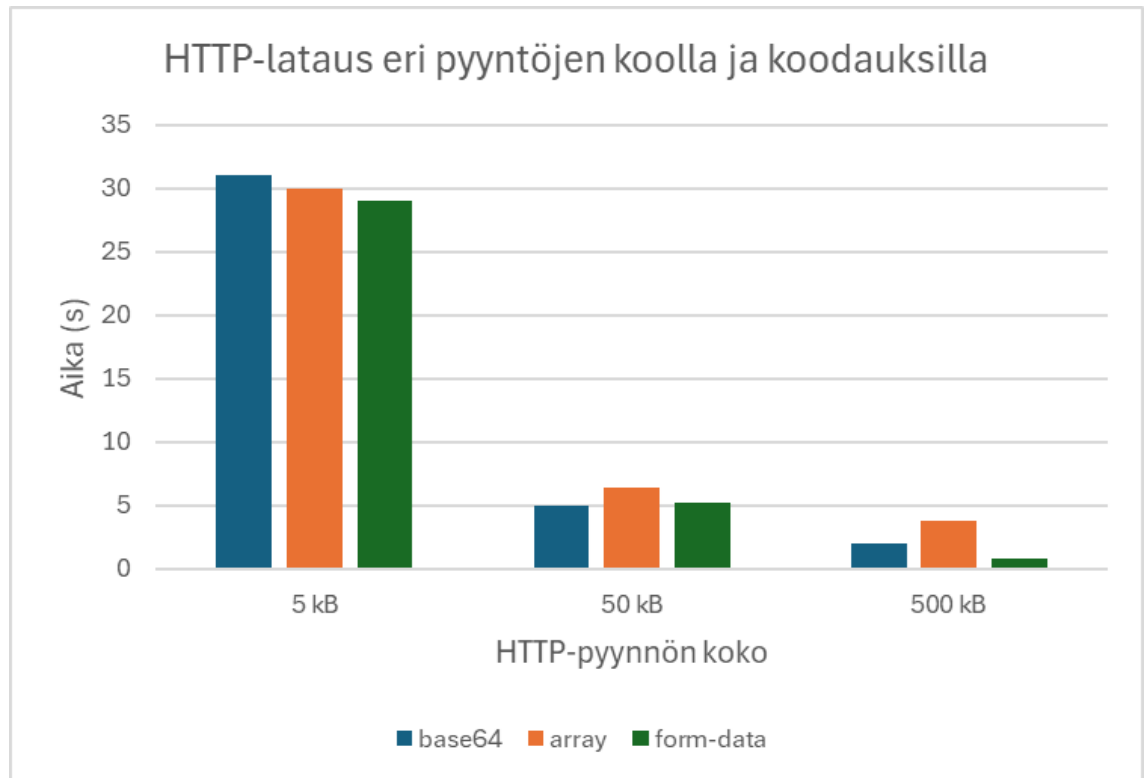


Kuva 17. FTP-, FTPS- ja SFTP-latausten tulokset Python-koodi.



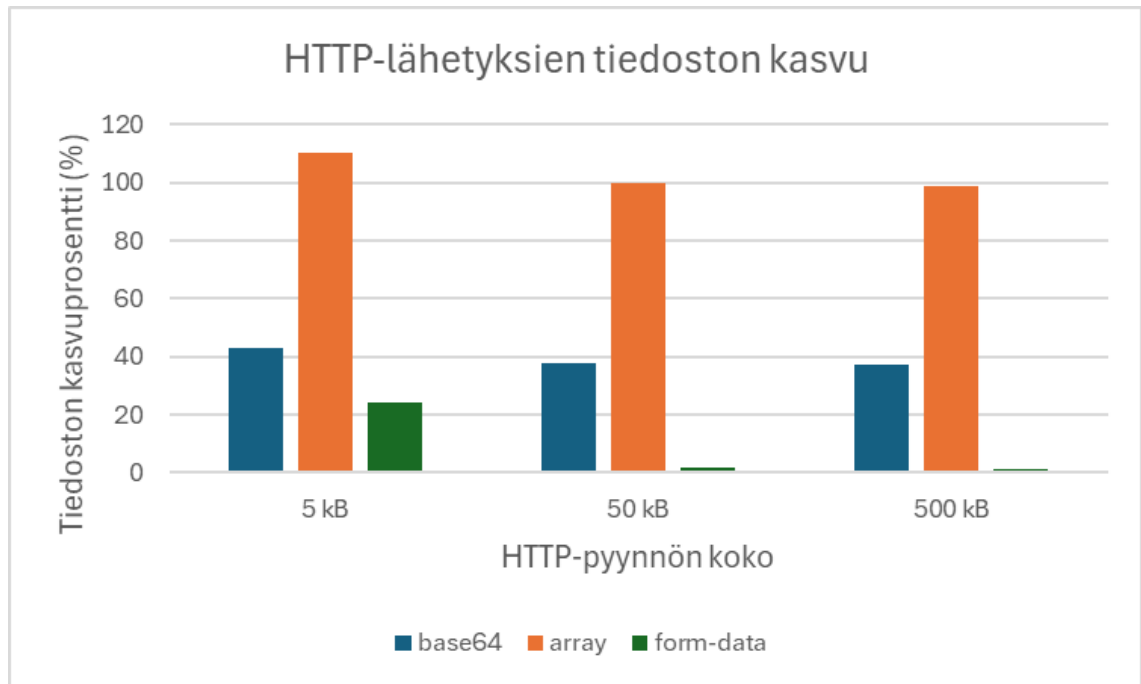
Kuva 18. FTP-, FTPS- ja SFTP-latausten tulokset C#-koodi.

Kuva 19 esittää HTTP-latausten latausajat erikokoisilla pyyntöjen koolla ja koodauksilla. Lataus suoritettiin käyttöliittymän kautta, jossa valittiin koodaus ja pyynnön koko. Käyttöliittymä luki lähetettävän tiedoston osissa ja lähetti nämä osat palvelimelle. Pynnön koko vastasi luetun tiedoston osan kokoa. Kuvasta näkee selvästi, että pyynnön koon kasvaessa latausaika pienenee merkittävästi kaikilla koodaustavoilla. Pienempi pyynnön koko aiheuttaa useampia pyyntöjä palvelimelle, mikä hidastaa latausta huomattavasti. 5 kB:n kokoisilla pyynnöillä hitain koodaustapa on base64, vaikka teoriassa array olisi hitain, koska se kasvattaa eniten siirrettävän datan kokoa.



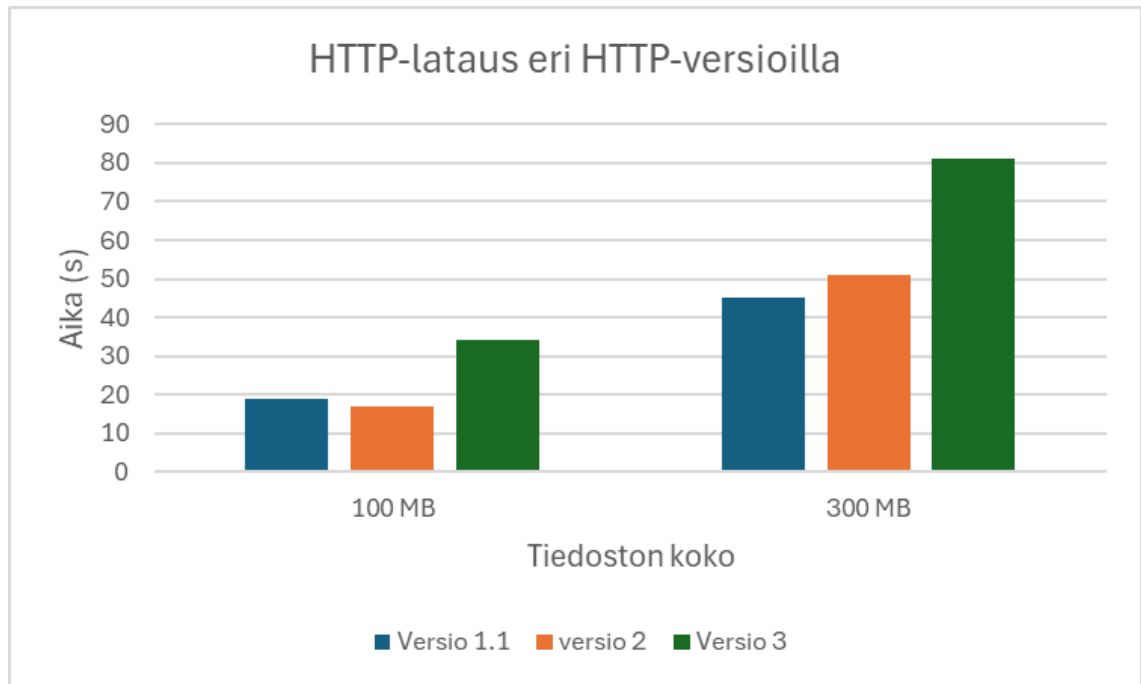
Kuva 19. HTTP-latausten latausajat.

Kuva 20 esittää HTTP-latausten yhteydessä lähetettävän datan määrän suhteessa tiedoston kokoon. Lähetettävä data on mitattu Wireshark-ohjelmalla. Siirrettävässä datassa on mukana kaikki TCP-tasolla siirtynyt data, joten se kattaa sekä siirrettävän tiedoston että HTTP-pyyntöön tarvittavan datan. Kun käytetään pientä, 5 kB:n kokoista pyyntöä, lähetettävän datan määrä kasvaa merkittävästi kaikilla koodaustavoilla. Teoriassa form-data on kaikista tehokkain tapa lähettää dataa, sillä se siirtyy binäärimuodossa. Form-data kasvattaa lähetettävän datan määrää vain muutamalla prosentilla, kun pyynnön koko on suuri. Base64-koodaus kasvattaa lähetettävän tiedoston kokoa noin 33 %, kun käytetään suurta pyyntöä. Array-koodaus kasvattaa siirrettävän datan määrää jopa 100 %, mikä tekee siitä ehdottomasti huonoimman tavan siirtää tiedostoja HTTP-protokollan avulla. Ainoastaan 500 kB:n kohdalla on selvästi nähtävissä teoreettinen tulos, että form-data kasvattaa tiedostoa vähiten, base64-koodaus noin 33 % ja array kaikista eniten.



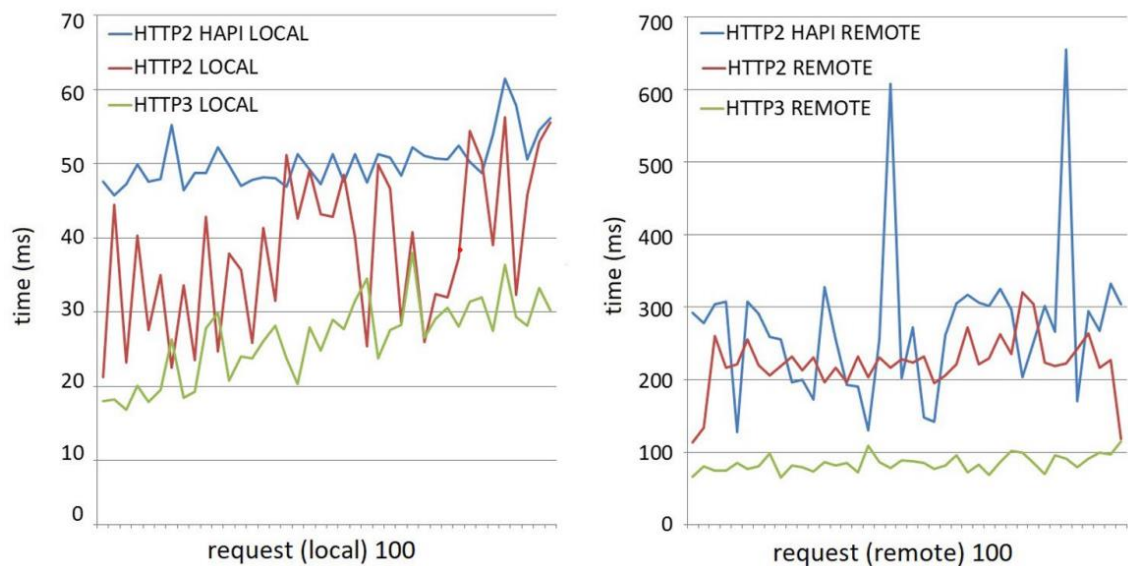
Kuva 20. HTTP-latausten tiedoston kasvu lähettäessä.

Kuva 21 esittää tuloksia HTTP-latauksista eri HTTP-versioilla. Suorituskykytestissä ovat mukana HTTP:n kolme uusinta versiota, ja siinä on käytetty kahta eri kokoista tiedostoa. Asiakasohjelmanä on käytetty C#-ohjelmointikielen HTTP-kirjastoa. Tuloksista nähdään, että HTTP/3-versio on hitain kummallakin tiedostokoolla. Projektissa on käytetty .NET 7 -versiota, joka oli .NET:n ensimmäinen versio, joka tuki HTTP/3-protokollaa. HTTP/3:n huono suorituskyky voi johtua siitä, että C#-koodi ei ole optimoitu kunnolla tälle protokollalle. HTTP/1.1 ja HTTP/2 ovat suorituskyvyiltään lähes samalla tasolla; HTTP/1.1 on nopeampi suuremmalla tiedostolla, kun taas HTTP/2 on nopeampi pienemmällä tiedostolla. Teorian mukaan HTTP/2:n pitäisi olla nopeampi binäärisen siirtokerroksen ansiosta.



Kuva 21. HTTP-lataus eri HTTP-versioilla.

Kun HTTP-versioiden suorituskyvyn tuloksia verrataan aikaisempiin tutkimustuloksiin, huomataan merkittävä ero. Kuva 22 esittää Heidelbergin yliopistollisen sairaalan tekemän tutkimuksen HTTP/2- ja HTTP/3-versioiden suorituskyvystä vuodelta 2024. Kuva on otettu suoraan tutkimuksesta.

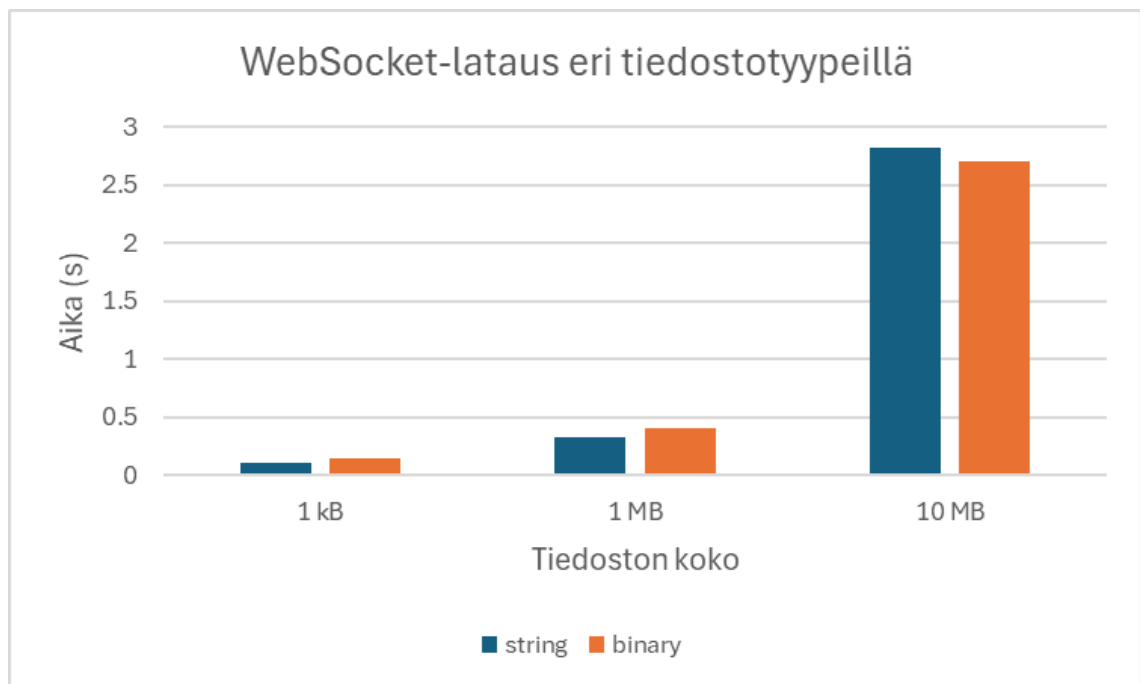


Kuva 22. Heidelbergin yliopistollisen sairaalan tekemä HTTP/2- ja HTTP/3-versioiden vertailu vuodelta 2024.[74]

Tutkimuksessa vertailtiin protokollien suorituskykyä sekä sisäverkossa että ulko-verkossa. Vasemmanpuoleinen kuvaaja esittää sisäverkon tuloksia, ja oikeanpuoleinen kuvaaja esittää ulko-verkon tuloksia. Tutkimuksessa käytettiin Tomcat

HAPI -palvelinta HTTP/2-protokollan kanssa sekä Hypercorn-palvelinta HTTP/2- ja HTTP/3-protokollien kanssa. Kuvaajissa on esitetty sadan yksittäisen HTTP-pyynnön käyttämä aika. Tutkimuksen mukaan HTTP/3 on huomattavasti nopeampi kuin HTTP/2 sekä sisäverkossa että ulkoverkossa. Tulokset ovat yhdenmukaisia teorian kanssa, sillä HTTP/3 kehitettiin tarjoamaan parempi suorituskyky kuin HTTP/2.[74]

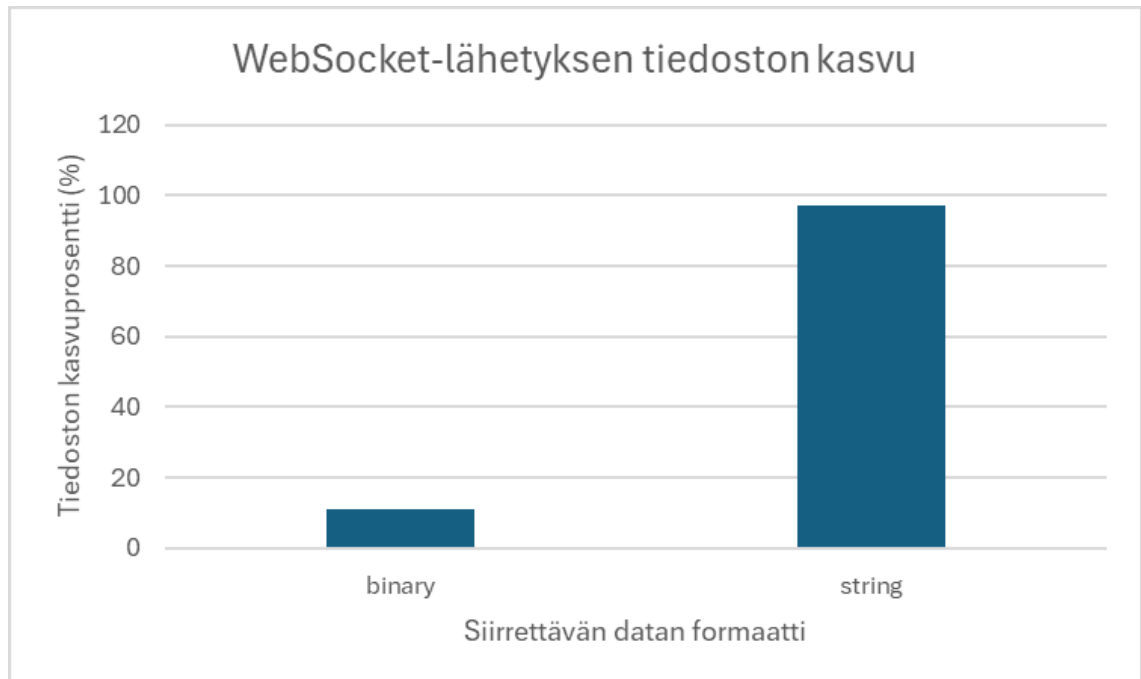
Kuva 23 esittää WebSocket-latauksen tulokset. WebSocket-yhteys luotiin .NET:n SignalR-kirjaston avulla. SignalR on osa .NET:iä, joten sitä ei tarvitse erikseen asentaa palvelimelle, mutta selaimeen se piti asentaa npm-pakettina. SignalR käyttää selaimen WebSocket API:a konepellin alla, mutta se luo korkeamman tason rajapinnan, mikä mahdollistaa palvelinpään funktioiden kutsumisen käyttööliittymästä WebSocket-yhteyden avulla. Binääriviestintää varten SignalR-kirjastoon täytyy asentaa lisäosa, MessagePack, NuGet-pakettina, sekä käyttööliittymään saman niminen npm-paketti. Lataus suoritettiin sekä merkkijono- että binäärimuodossa, ja testissä käytettiin kolmea erikokoista tiedostoa. Latausajat olivat hyvin samanlaiset riippumatta siitä, käytettiinkö merkkijono- vai binääriformaattia. Pienimmillä tiedostoilla merkkijono oli hieman nopeampi, mutta suuremmilla tiedostoilla binääri oli hieman nopeampi. Teoriassa binääriformaatin pitäisi olla nopeampi, koska siirrettävän datan määrä on pienempi.



Kuva 23. Websocket-lataus eri kokoisilla tiedostoilla.

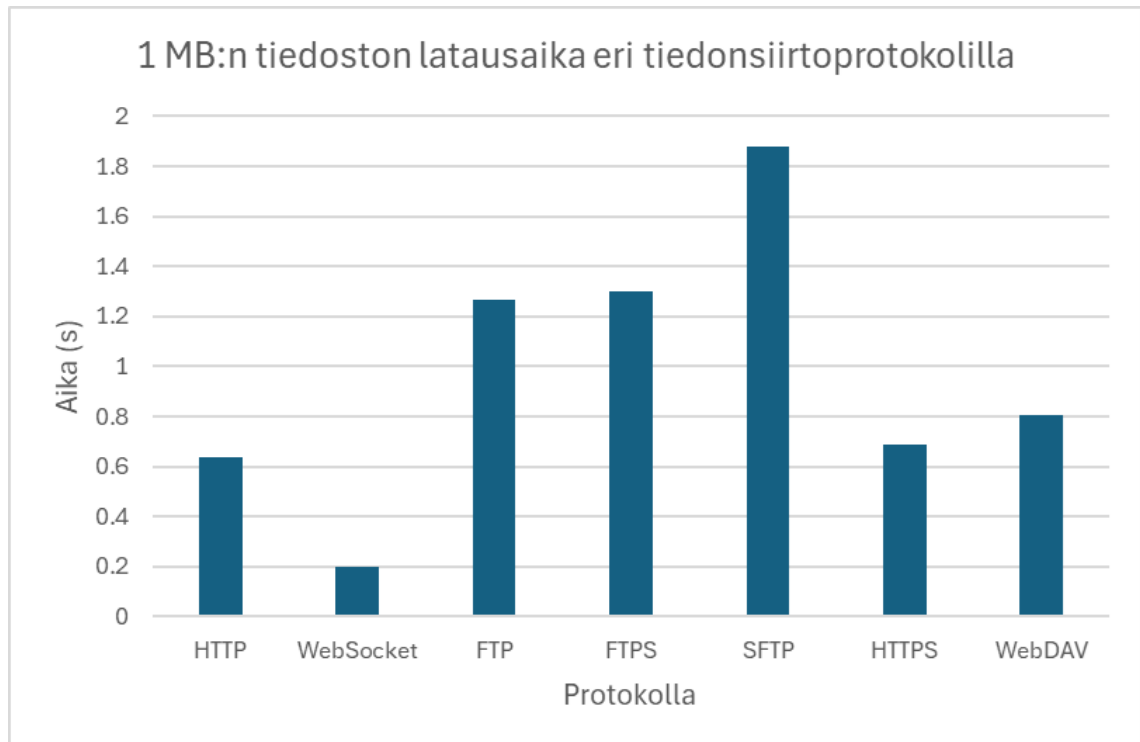
Kuva 24 esittää siirrettävän datan määrän suhteessa alkuperäisen tiedoston kokoon. Siirrettävä data on mitattu TCP-tasolla, samalla tavalla kuin HTTP-pyyntöjen siirrettävän datan määrää vertailtaessa. String-formaatissa lähetettäessä siirrettävän

datan määrä on lähes kaksinkertainen alkuperäiseen tiedostoon nähden, kun taas binary-formaatissa siirrettävä data on 11 % suurempi kuin alkuperäinen tiedosto.



Kuva 24. WebSocket-latauksessa siirrettävän datan kasvu.

Kuva 25 esittää tiedostojen siirtoaikoja eri siirtoprotokollilla. Suorituskykytestissä siirrettiin 1 MB:n kokoinen tiedosto kaikilla tutkituilla protokollilla, ja kaikissa käytettiin C#-kielen kirjastoja. Tuloksista havaitaan samanlaisia tuloksia kuin aiemmissa vertailuissa: SFTP on selvästi hitain protokolla, kun taas FTP ja FTPS ovat lähes saman nopeuksisia. Vastaavasti HTTP ja HTTPS ovat myös lähes saman nopeuksisia. WebDav on hieman hitaampi kuin HTTP, vaikka molemmat lähettävät tiedoston HTTP-pyyntönä. WebDav-palvelimena käytettiin Apachea ja HTTP-palvelimena .NET:iä, mikä saattaa selittää, miksi WebDav on hitaampi kuin HTTP. Ylivoimaisesti nopein protokolla tutkimuksessa oli WebSocket.

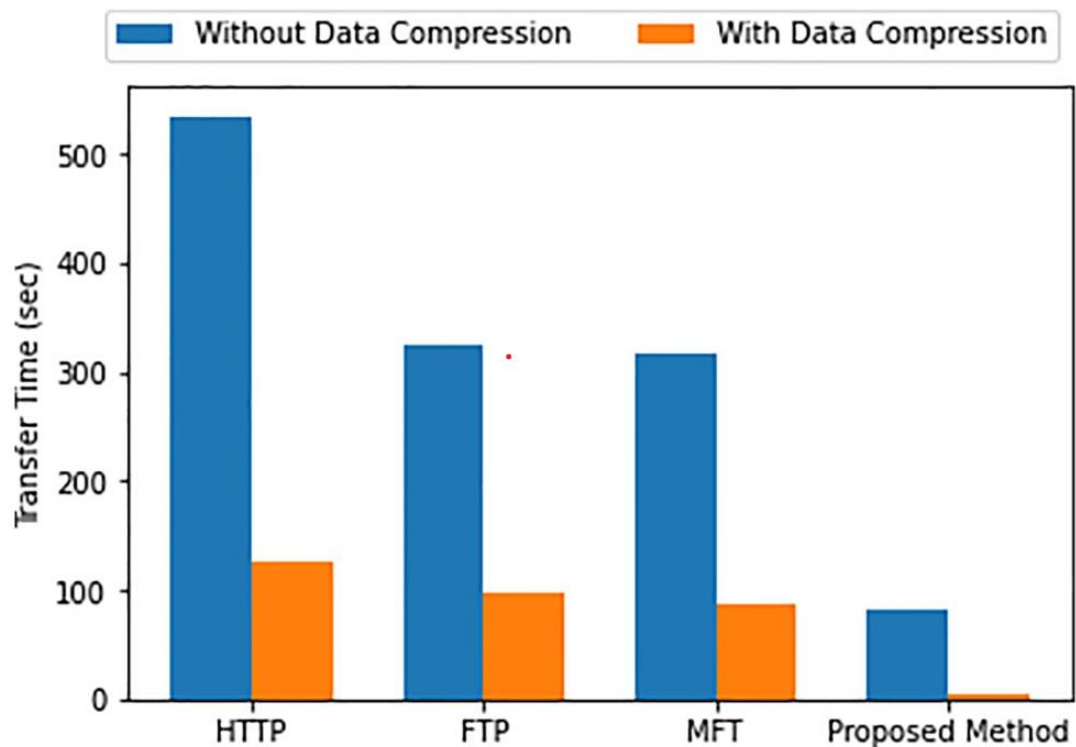


Kuva 25. Tiedoston siirtoaika eri protokollilla.

Kun vertaillaan tässä työssä saatuja tuloksia eri protokollien siirtoajoista aikaisempiin tutkimustuloksiin, huomataan pieniä eroja. Kuva 26 esittää Kaurin, Chauhanin ja Aggarwalin vuonna 2022 tekemän tutkimuksen tuloksia, jossa tutkittiin FASTA-tiedostojen siirtoa eri tiedonsiirtotavoilla. Kuva on otettu suoraan tutkimuksesta. FASTA-tiedostot ovat tekstipohjainen formaatti nukleotidi- tai aminohapposekvenssien esittämiseen. Tutkimuksessa käytetyt tiedonsiirtotavat olivat HTTP, FTP, MFT sekä tutkijoiden kehittämä koneoppimiseen perustuva algoritmi, joka pakkaa FASTA-tiedostoja. Managed File Transfer (MFT) tarjoaa hallintatyökaluja tiedonsiirtojen seuraamiseen, hallintaan ja automatisointiin sekä mahdollistaa useiden tiedonsiirtoprotokollien käytön. Tutkimuksessa MFT:n osalta käytettiin FTPS-protokollaa. Siirrettävä tiedosto oli kooltaan 500 MB, ja siirto tehtiin sekä ilman että tiedonsiirtotapojen tarjoamaa pakkausta hyödyntäen, ja siirtoaika mitattiin. Tutkimuksen tulokset osoittavat, että FTP- ja FTPS-protokollien suorituskyky on lähes identtinen, mutta HTTP-protokollan suorituskyky on heikompi sekä pakattuna että pakkaamattomana.[75]

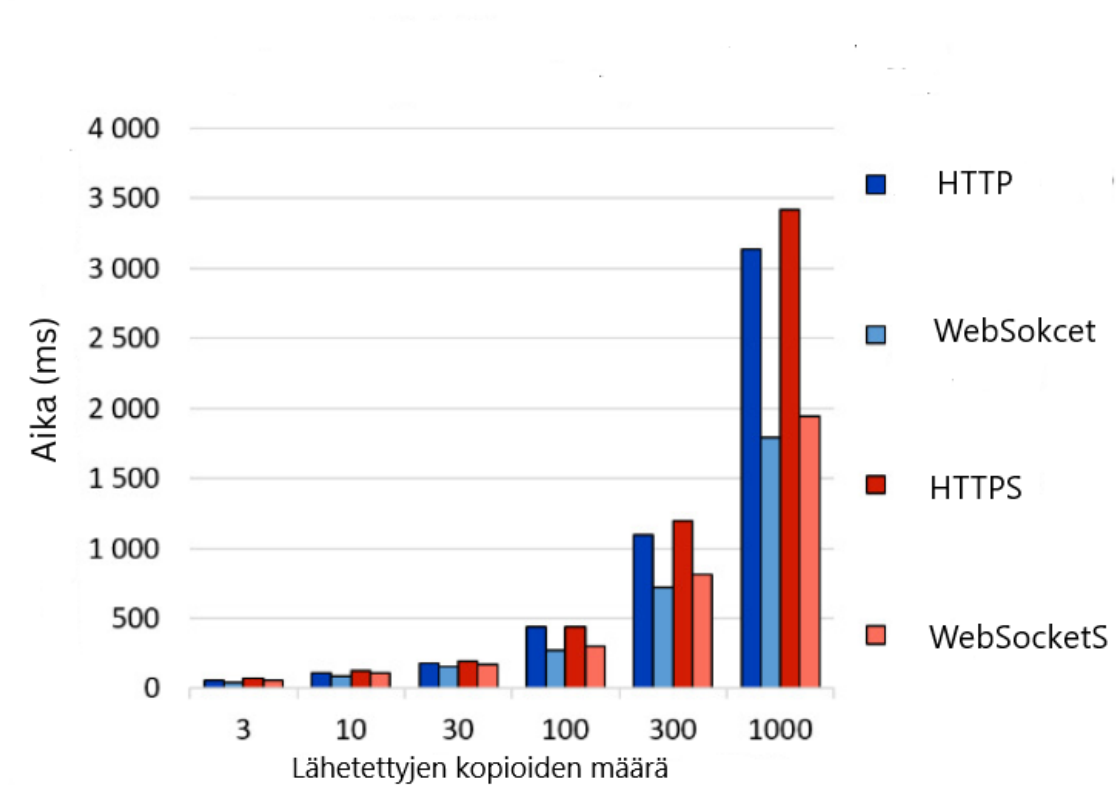
Tämän työn tulokset poikkeavat hieman Kaurin, Chauhanin ja Aggarwalin tekemästä tutkimuksesta. Kuva 25 esittää tämän työn tuloksia, joiden perusteella HTTP:n suorituskyky on parempi kuin FTP- ja FTPS-protokollien, joiden suorituskyky on lähes identtinen. Yksi mahdollinen syy HTTP-protokollan suorituskykyeroihin tutkimuksissa

voi olla käytetty HTTP-palvelin ja ohjelmointikieli. Tässä työssä käytetty .NET tarjoaa yhden parhaista HTTP-palvelinten suorituskyvyistä.[75]



Kuva 26. Kaurin, Chauhan ja Aggarwalin tekemän tutkimuksen tulokset protokollien vertailusta vuodelta 2022.[75]

Lublinin teknillinen yliopisto tutki HTTP-, HTTPS-, WebSocket- ja WebSocketS-protokollien suorituskykyä vuonna 2021. Tutkimuksessa palvelimen ja selaimen välillä siirrettiin 100-merkkistä tekstitiedostoa. Tekstitiedostosta lähetettiin useita kopioita yhtäaikaaisesti eri protokollilla, ja tutkittiin, miten lähetettyjen kopioiden määrä vaikuttaa suorituskykyyn. Kopiot lähetettiin yhden WebSocket-yhteyden avulla, mutta jokaiselle kopiolla tarvittiin oma HTTP-pyyntö. Kopioiden määrä vaihteli 3:n ja 3000:n välillä eri testeissä. Kuva 27 esittää tutkimuksen tuloksia, kun kopioiden määrä oli 3–1000, ja siirtoon käytetty aika mitattiin. Kuva on otettu tutkimuksesta, mutta siihen on vaihdettu puolankieliset tekstit suomenkielisiksi. Kuvasta huomataan, että TLS-salauksen lisääminen vaikuttaa suorituskykyyn erittäin vähän kummassakin tutkitussa protokollassa. Rinnakkaisten tiedostokopioiden määrän kasvattaminen lisää merkittävästi suorituskykyeroa HTTP- ja WebSocket-protokollien välillä. Kun kopioiden määrä on 1000, HTTP-protokollan käyttämä aika on lähes 100 % suurempi kuin WebSocketin käyttämä aika. Pienemmillä kopiomäärillä suorituskyvyn ero pienenee, mutta WebSocket on silti nopeampi myös pienimmällä kopioiden määrällä.[76]



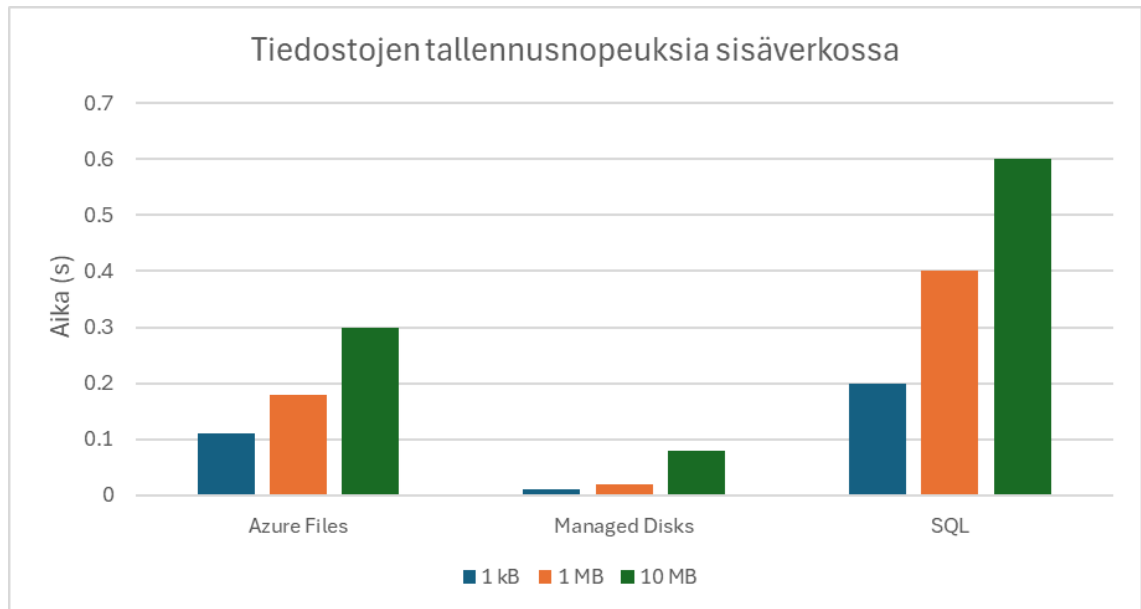
Kuva 27. Lublinin teknillisen yliopiston tekemä tutkimus HTTP(S)- ja WebSocket(S)-protokollien suorituskyvystä vuodelta 2021.[76]

Lublinin teknillisen yliopiston saamat tulokset eroavat hieman tässä työssä saaduista. Tässä työssä lähetettiin vain yhtä tiedostoa kerrallaan palvelimelle, ja tuloksena havaittiin, että WebSocket-protokolla on huomattavasti nopeampi kuin HTTP-protokolla. Teknillisen yliopiston tuloksissa WebSocket on vain hieman nopeampi kuin HTTP, kun rinnakkaisia lähetyksiä on vähän, mutta huomattavasti nopeampi, kun rinnakkaisia lähetyksiä on 1000.[76]

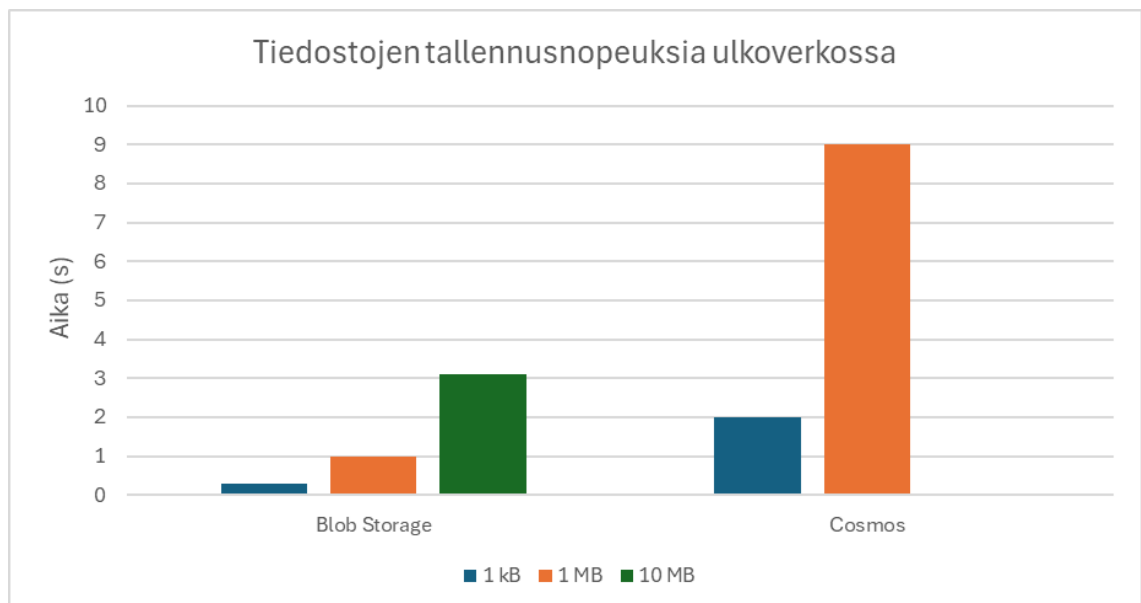
4.2.2 Tiedostojen tallennus

Kuva 28 esittää tiedostojen tallennusnopeudet sisäverkon tallennusratkaisuilla ja Kuva 29 esittää julkisen verkon tallennusratkaisuiden tallennusnopeudet. Kaikissa tallennusratkaisuissa on käytetty kylmää palvelumallia. Kuvista huomataan, että Managed Disks ja Azure Files ovat huomattavasti nopeimpia tallennusratkaisuja. Tämä johtuu siitä, että niihin vaikuttavat sisäverkon verkkoliikenne, eivätkä ne vaadi kirjautumismekanismeja. SQL-tallennus on seuraavaksi nopein, sillä data siirretään TCP-yhteyden kautta sisäverkossa. Kuitenkin SQL-kyselyt edellyttävät kirjautumis- ja datan eheysprosesseja, jotka hidastavat tallennusta. Blob Storage ja Cosmos-tietokanta käyttävät julkisen verkon HTTP API:a tiedostojen tallentamiseen, mikä tekee

niistä ehdottomasti hitaimpia tallennusvaihtoehtoja. Blob Storage mahdollistaa binääritiedostojen tallentamisen binäärimuodossa, mutta Cosmos-tietokanta käyttää pelkästään JSON-dokumentteja, jolloin tiedostojen koko kasvaa merkittävästi, kun ne tallennetaan tekstimuodossa. Lisäksi Cosmos-tietokannan JSON-dokumentin maksimikoko on 2 MB, minkä vuoksi Cosmos-tietokannan suorituskykyä ei voitu testata 10 MB:n tiedostoilla.



Kuva 28. Sisäverkossa olevien tallennusratkaisuiden tallennusnopeuksia.

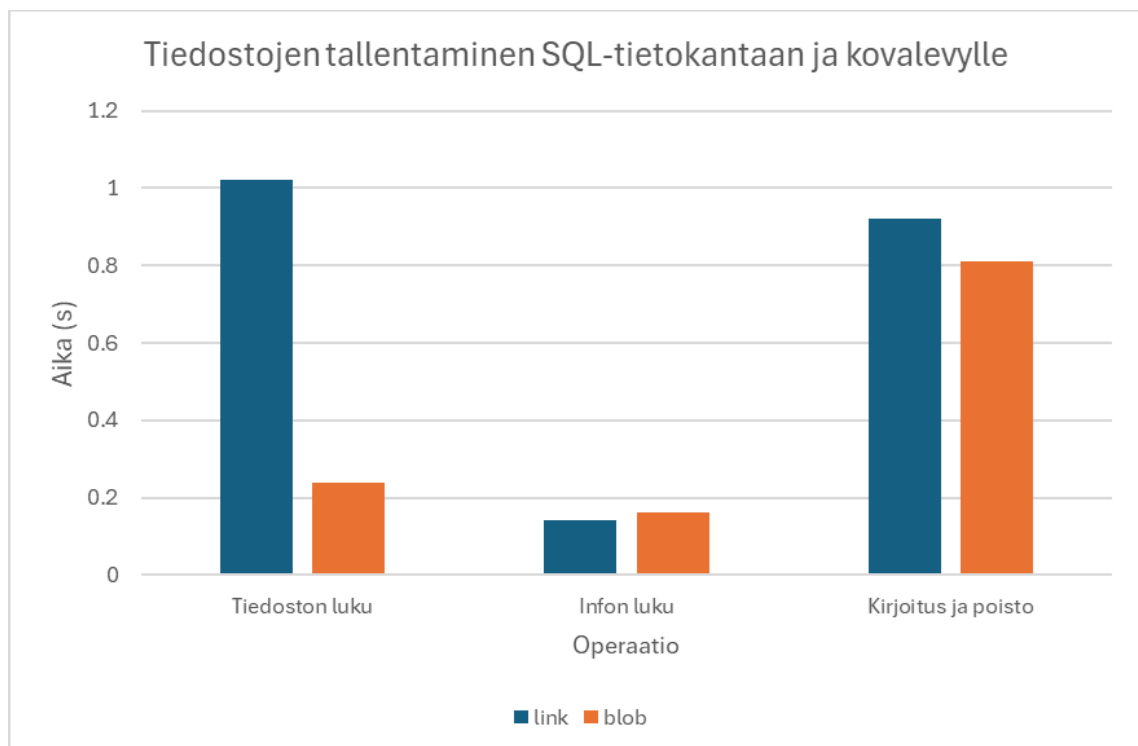


Kuva 29. Ulkoverkossa olevien tallennusratkaisuiden tallennusnopeuksia.

Yksi tutkimuksista vertaili tiedostojen tallentamista SQL-kantaan binäärisenä BLOB-tyyppinä. Toinen vaihtoehto on tallentaa SQL-tietokantaan tiedoston tiedostopolku ja

säilyttää itse tiedosto kovalevyllä tietokannan sijaan. Kuva 30 esittää tutkimuksen tulokset. Tutkimuksessa vertailtiin tiedostojen lukua, kirjoitusta, poistoa sekä tiedostoinfon lukua tietokannasta. Vertailussa käytettiin kahta tietokantataulua, jotka ovat muuten identtiset, mutta toisessa tiedosto on tallennettu tietokantaan ja toisessa vain tiedoston linkki tiedostojärjestelmään. Tiedostoinfo on tallennettu samalle riville tiedoston tai sen linkin kanssa, ja tyypiltään se on varchar.

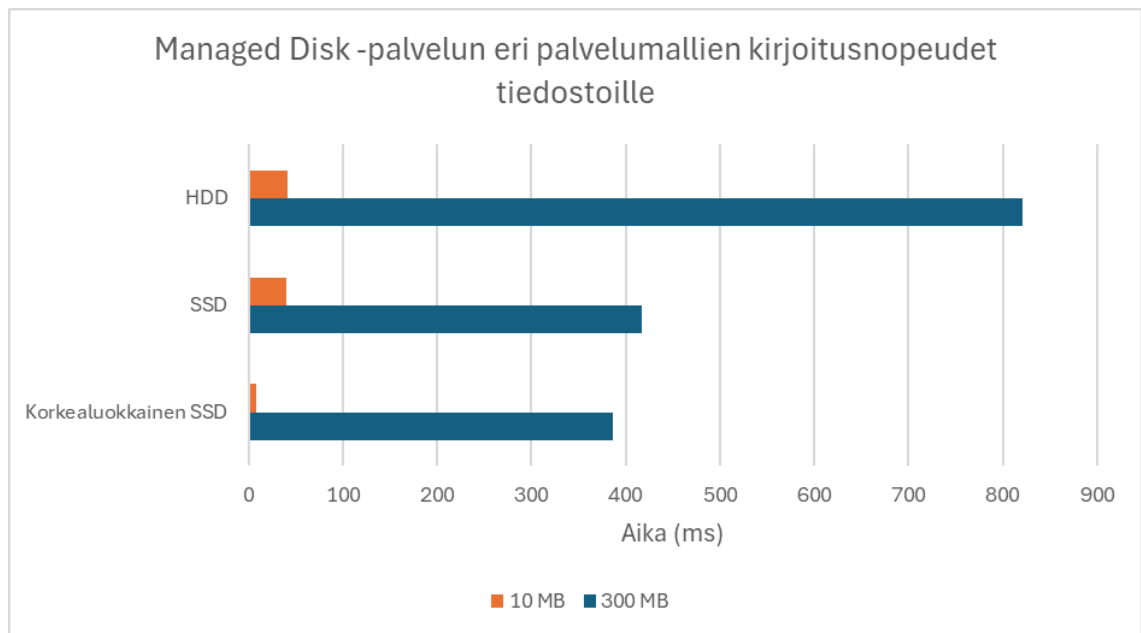
Tuloksista nähdään selvästi, että tiedoston luku on nopeampaa, kun tiedosto on tallennettu tietokantaan, koska erillistä levylukuoperaatiota ei tarvitse tehdä tietokantakyselyn jälkeen. Tiedostoinfon luku on puolestaan hieman nopeampaa taulussa, jossa on tiedostolinkki, koska rivien indeksointi on nopeampaa RAM-muistissa, sillä rivien koko on pienempi. Kirjoitus- ja poistonopeudet ovat lähes samat molemmissa tallennustavoissa. Yksi tärkeä huomio tiedostojen tallentamisessa tiedostojärjestelmään on tietokannan eheys. Kun tiedostot tallennetaan suoraan tietokantaan, tietokanta pysyy eheänä tiedostojen suhteen. Sen sijaan levyllä tallennettaessa eheys voi rikkoutua, koska tietokantaoperaatio ja levyoperaatio eivät ole samassa tietokantatransaktiossa.



Kuva 30. Tiedostojen tallentaminen SQL-tietokantaan ja kovalevyllä.

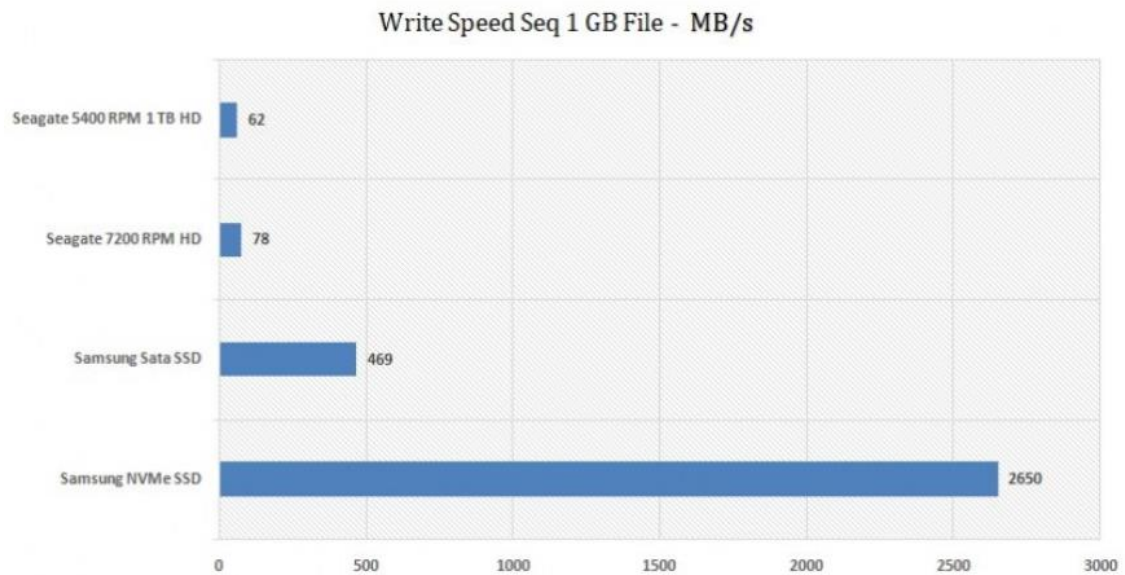
Seuraavaksi vertaillaan eri tallennusratkaisujen palvelumallien vaikutusta suorituskykyyn. Vertailtavina tallennusratkaisuinä ovat Managed Disks, Azure Files ja Blob Storage. Vertailu tehtiin kahden erikokoisen tiedoston avulla: 10 MB ja 300 MB. Molemmat tiedostot tallennettiin kaikilla palvelumalleilla 50 kertaa, ja jokaisen tiedoston

tallentamiseen käytetty aika mitattiin. Näistä ajoista laskettiin keskiarvo. Kuva 31 esittää Managed Disks -tallennusratkaisun tuloksia. Pienemmän, 10 MB:n tiedoston kohdalla HDD- ja SSD-levyjen tallennusaika on lähes identtinen, kun taas korkealuokkaisen SSD-levyn aika on noin viisi kertaa lyhyempi. Suuremman, 300 MB:n tiedoston kohdalla HDD on selvästi hitain ratkaisu, ja vaikka SSD- ja korkealuokkaisen SSD-levyn suorituskkyky on lähes sama, korkealuokkainen SSD on hieman nopeampi



Kuva 31. Managed Disks -tallennusratkaisun palvelumallien vertailu.

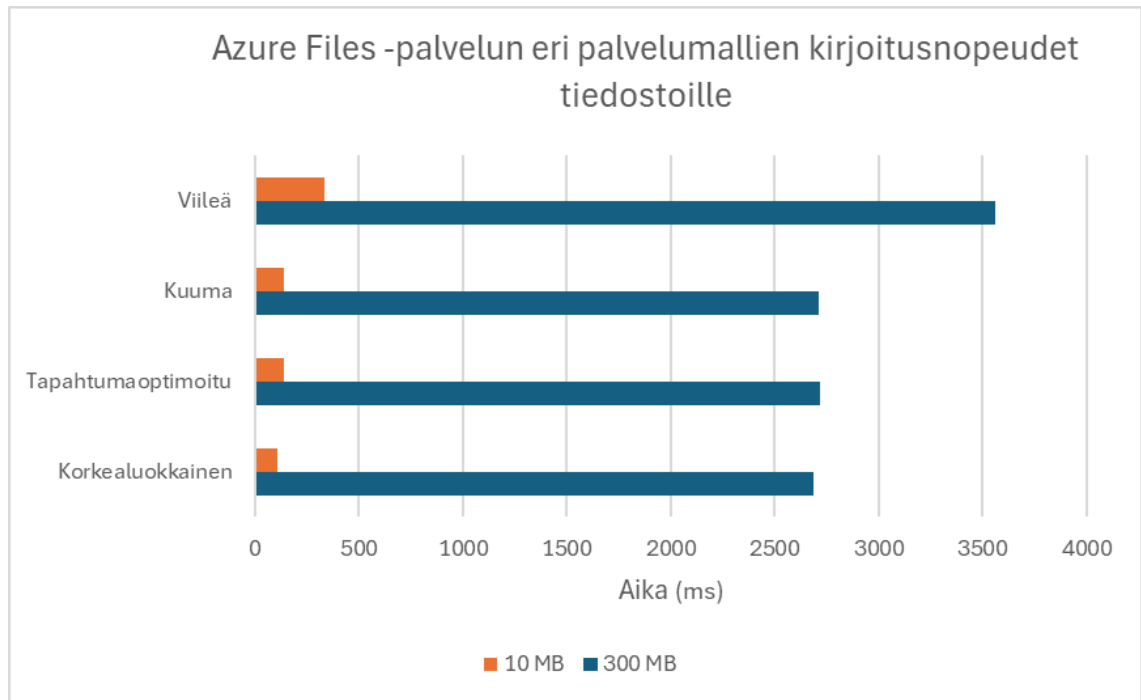
HDD- ja SSD-kiintolevyjen suorituskvyyistä on myös aiempia tutkimustuloksia. Kuva 32 esittää Candytechin tekemää tutkimusta HDD- ja SSD-kiintolevyjen suorituskvyyistä. Kuva on otettu suoraan tutkimuksesta. Tutkimuksessa käytettiin kahta eri HDD- ja kahta SSD-kiintolevyä, joille kirjoitettiin 1 GB:n kokoinen tiedosto. Tutkimuksessa mitattiin kiintolevyjen kirjoitusnopeus megatavuina sekunnissa (MB/s).[77]



Kuva 32. Candytechin tekemä tutkimus HDD- ja SSD-kiintolevyjen suorituskyvystä.[77]

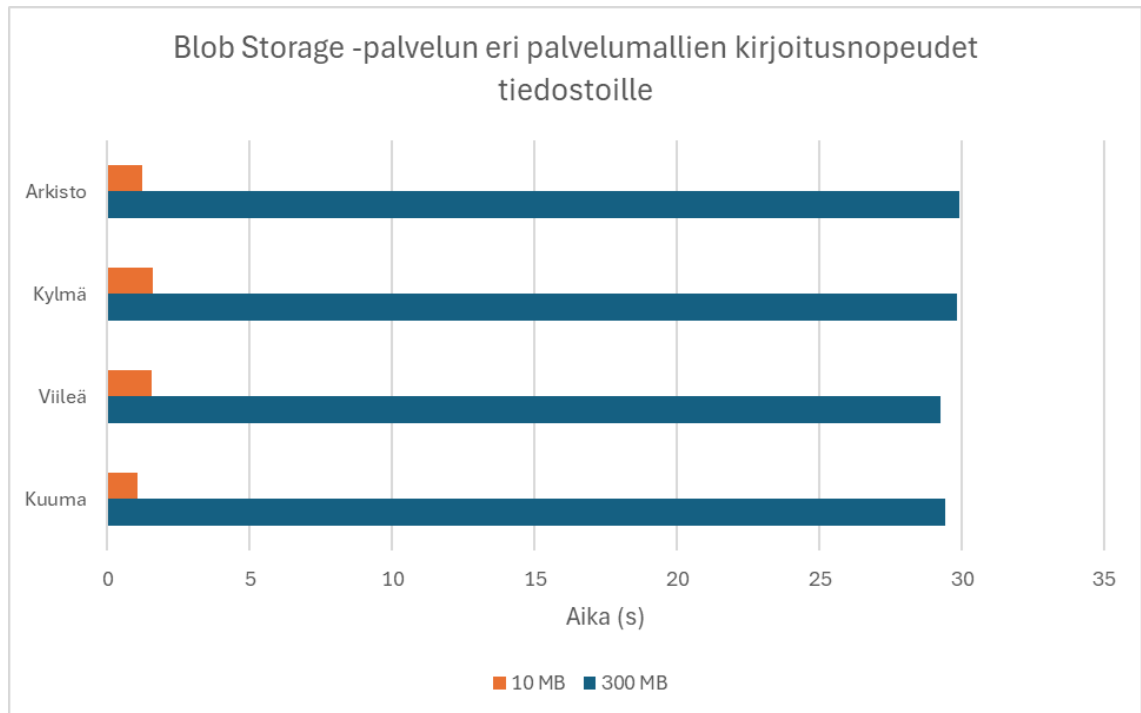
Candytechin tutkimuksessa ei havaittu eroja HDD-kiintolevyjen välillä, mutta SSD-kiintolevyjen välillä havaittiin merkittäviä eroja. Hitaamman SATA SSD -levyn kirjoitusnopeus oli noin 500 % suurempi kuin HDD-levyn. Nopeampi SSD-levy käytti Non-Volatile Memory Express (NVMe) -protokollaa, joka on uusi SSD-levyille suunniteltu ja huomattavasti nopeampi kuin SCSI-protokolla. Tutkimuksen mukaan NVMe SSD -levyn kirjoitusnopeus oli noin 460 % suurempi kuin SATA SSD -levyn. Myös tämän työn tulokset olivat samansuuntaiset Candytechin tutkimuksen kanssa: SSD-levyn kirjoitusnopeus suurella, 300 MB:n tiedostolla oli noin 100 % nopeampi kuin HDD-kiintolevyn. NVMe-protokollaa ei käsitelty tässä työssä, sillä vain harvat, erittäin kalliit Azuren virtuaalikoneet tukevat NVMe-levyjä. [77]

Kuva 33 esittää Azure Files -tallennusratkaisun tuloksia. 10 MB:n tiedoston kohdalla viileä tallennusratkaisu on yli kaksi kertaa hitaampi kuin kuuma tai tapahtumaoptimoitu ratkaisu, ja korkealuokkainen ratkaisu on noin 30 % nopeampi kuin tapahtumaoptimoitu. 300 MB:n tiedoston kohdalla viileä ratkaisu on noin 24 % hitaampi kuin muut palvelumallit, joiden tallennusnopeudet ovat lähes samat.



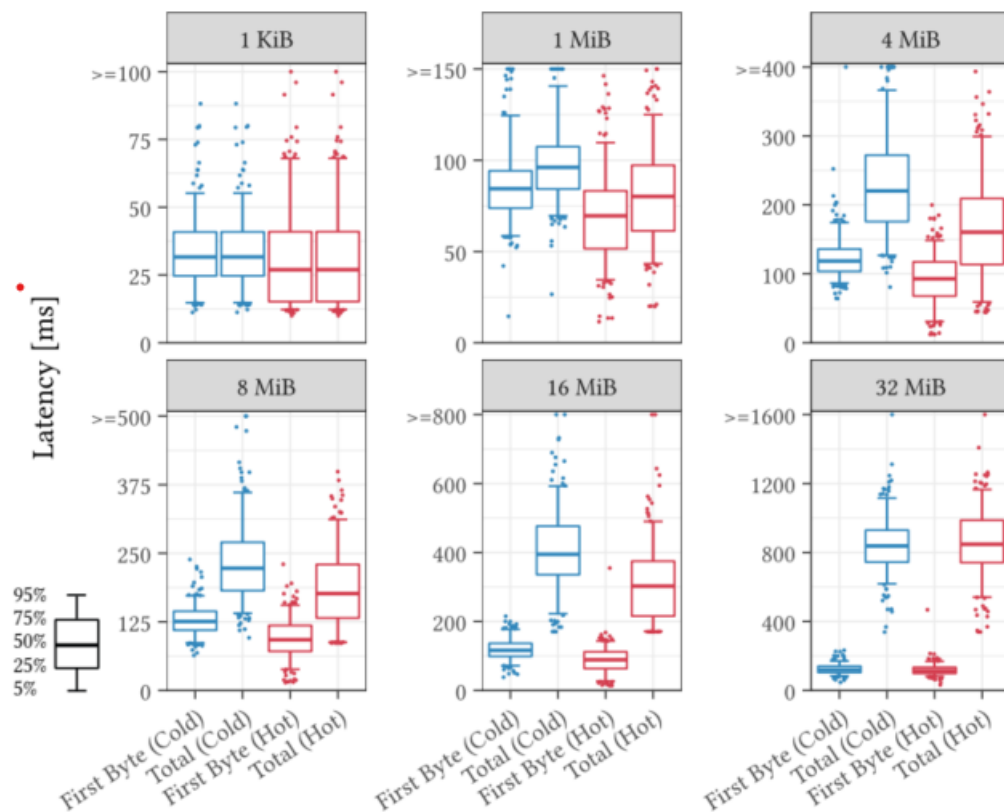
Kuva 33. Azure Files tallennusratkaisun palvelumallien vertailu.

Kuva 34 esittää Blob Storage -tallennusratkaisun tuloksia. Blob Storagea käytettiin julkisen HTTP-rajapinnan kautta, joten latausajat olivat riippuvaisia internetyhteyden nopeudesta. Tiedoston siirtämiseen kuluva aika oli merkittävä tekijä tässä tutkimuksessa. Tämän seurauksena palvelumalleilla ei ollut vaikutusta 300 MB:n tiedoston tallentamiseen, ja 10 MB:n tiedoston tallennusaikojen erot johtuivat todennäköisesti internetyhteyden nopeuden hetkellisistä vaihteluista. Arkisto-palvelumallista ei voi ladata tiedostoja suoraan, vaan tiedostot täytyy ensin siirtää toiseen palvelumalliin. Tämän palvelumallin muutos voi pahimmassa tapauksessa kestää useita tunteja.



Kuva 34. *Blob Storage -tallennusratkaisun palvelumallien vertailu.*

Samanlaisia tuloksia Blob Storagen palvelumallien suorituskyvystä on saatu myös aikaisemmissa tutkimuksissa. Kuva 35 esittää Durnerin, Dominikin ja Leisin vuonna 2023 tekemän tutkimuksen tuloksia eri pilvipalveluiden Object storagien suorituskyvystä. Kuva on otettu suoraan tutkimuksesta. Kuvassa tarkastellaan AWS-pilvipalvelun S3-palvelun kylmän ja kuuman palvelumallin suorituskykyä tiedostojen lataamisessa eri kokoisilla tiedostoilla. Kuvassa esitetään sekä koko lataukseen kuluva aika että ensimmäisen tavun lataamiseen kuluva aika kuudella eri tiedostokoolla.[78]

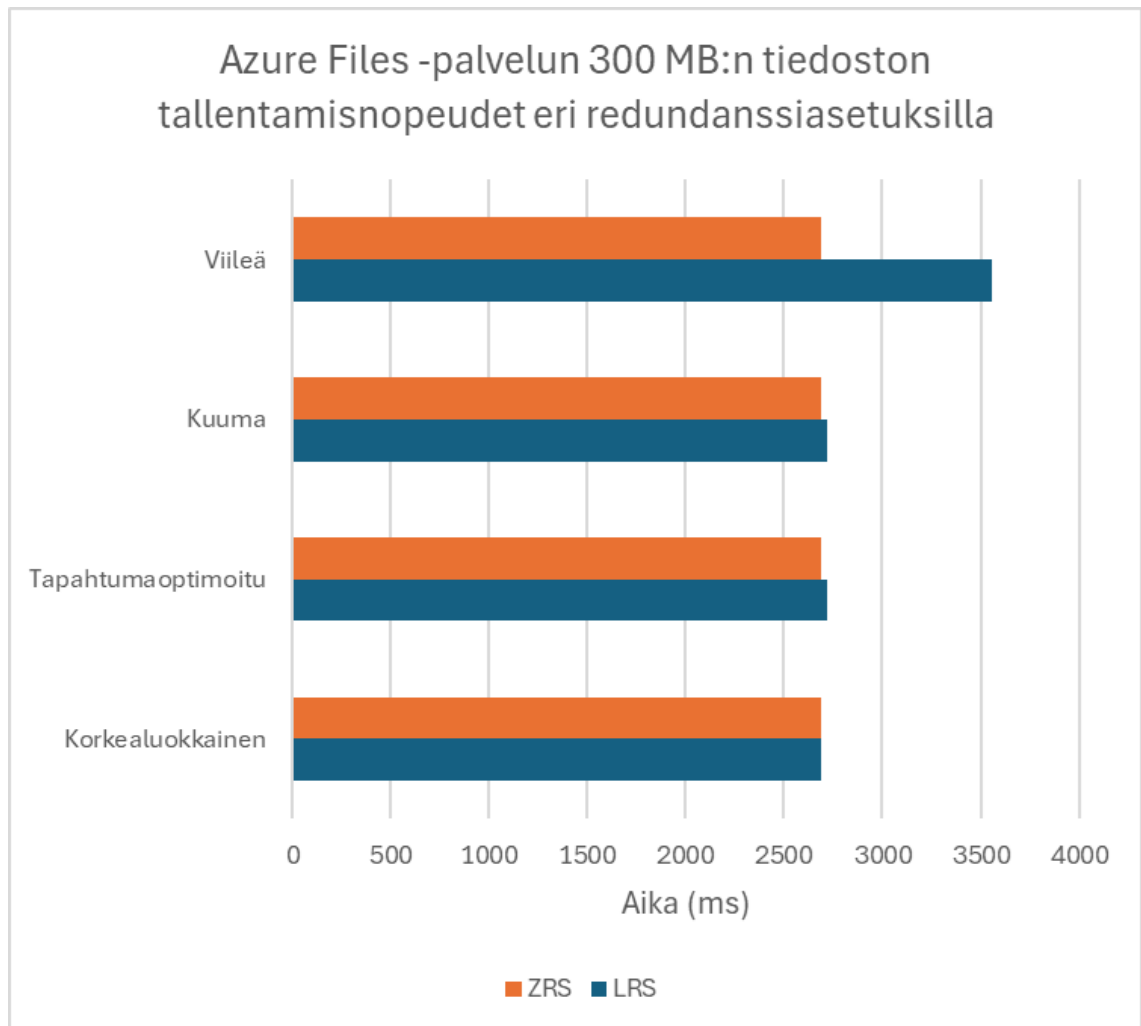


Kuva 35. AWS-pilvipalvelun S3:n kylmän ja kuuman palvelumallin suorituskyky Durnerin, Dominikin ja Leisin tekemässä tutkimuksessa vuonna 2023.[78]

Durnerin, Dominikin ja Leisin tutkimuksesta havaitaan, että kuuma palvelumalli on hieman nopeampi pienillä tiedostoilla kuin kylmä palvelumalli. Lisäksi pienillä tiedostoilla ensimmäisen tavun latausaika on lähes sama kuin koko tiedoston latausaika. Suuremmilla tiedostoilla koko tiedoston lataaminen vie huomattavasti enemmän aikaa kuin ensimmäisen tavun lataaminen. Suurimmalla, 32 MiB:n tiedostolla, kylmä palvelumalli on keskimäärin yhtä nopea kuin kuuma palvelumalli. Myös tässä työssä saatiin vastaavat tulokset: pienillä tiedostoilla kuuma palvelumalli oli hieman nopeampi kuin kylmä, mutta suurilla tiedostoilla palvelumallien suorituskyky oli lähes identtinen.[78]

Testeissä havaittiin, että suuremmalla tiedostolla kaikkien kuumien palvelumallien suorituskyky oli lähes sama. Testien aikana tallennusratkaisuissa ei tapahtunut muita I/O-operaatioita. Tulokset saattaisivat olla erilaiset, jos tallennusratkaisuihin kohdistettaisiin runsaasti I/O-operaatioita, sillä kuumimmat ratkaisut mahdollistavat suurimmat IOPS-suorituskyvyt.

Lopuksi vertaillaan vielä tallennustilin redundanssiasetuksen vaikutusta suoritussykyyn. Testissä vertailtiin Azure Files -tallennusratkaisun suoritussyky kirjoittaa 300 MB tiedosto paikallisesti redundantti tallennuksella (LRS) sekä Alueellinen redundantti tallennuksella (ZRS). Kuva 36 esittää testin tulokset. Redundanssiasetuksella ei ollut vaikutusta kuumilla tiedostoilla ja viileällä tiedostolla ZRS oli huomattavasti nopeampi.



Kuva 36. Redundanssiasetuksen vaikutus Azure Files -palveluun.

4.3 Tietoturva vertailu

Tässä luvussa esitellään suojaamattomien tiedonsiirtoprotokollien tietoturvaongelmat sekä vertaillaan suojattujen protokollien tietoturvaeroja. TLS-salauksen eri versioiden tietoturva haavoittuvuudet käydään läpi, ja esitetään tutkimustuloksia TLS-haavoittuvuuksien ja versioiden yleisyydestä. Lisäksi tarkastellaan Azuren pääsynhallintaa ja muutamia haavoittuvuuksia, joita on havaittu Azuressa. Lopuksi käsitellään muutamia tutkimuksia pilvipalveluihin kehitettyjen sovellusten tietoturvasta.

4.3.1 Suojaamattomat protokollat

Tietoturvavertailussa arvioidaan eri tallennustapojen ja tiedonsiirtoprotokollien tietoturvaa. Käsitellyistä tiedonsiirtotavoista suojaamattomia protokollia ovat HTTP, FTP, WebSocket ja WebDav. Näiden käyttöä ei nykyään suositella lainkaan, vaan niiden tilalla tulisi käyttää suojattuja versioita: HTTPS, FTPS, WebSocketS ja WebDavS. Suojaamattomat protokollat eivät salaa tiedonsiirtoa palvelimen ja asiakkaan välillä, mikä tarkoittaa, että kuka tahansa, joka pääsee verkkoliikenteeseen käsiksi, voi siepata ja lukea tiedot.[79][80]

Lisäksi suojaamattomat protokollat eivät tarjoa todennusta, mikä tekee niistä haavoittuvia välimieshyökkäyksille (MITM). MITM-hyökkäys on tilanne, jossa hyökkääjä pääsee kuuntelemaan ja muokkaamaan kahden käyttäjän välistä keskustelua. Hyökkäyksen tarkoituksena voi olla salakuunnella tietoja tai esiintyä toisena osapuolena manipuloimalla keskustelua. Tämä saa osapuolet uskomaan, että he kommunikoivat suoraan keskenään, vaikka hyökkääjä on asettunut heidän väliinsä.[79][80]

HTTP-protokollassa on lisäksi haavoittuvuuksia, joita ei voida torjua pelkällä TLS-salauksella. Yksi näistä on BREACH-haavoittuvuus, joka johtuu HTTP-protokollan pakkausalgoritmista. HTTP-pakkaus tehdään tavallisesti deflate-algoritmilla, jossa toistuvat tavusekvenssit havaitaan ja tallennetaan vain kerran, viitaten osoittimilla niiden sijaintiin. Tämä vähentää lähetettävien tavujen määrää. Kuitenkin pakatun datan pituus on nähtävissä myös salatussa muodossa, mikä mahdollistaa BREACH-hyökkäyksen.[81]

Sovellus on haavoittuvainen BREACH-hyökkäykselle, jos palvelimen HTTP-pakkaus on käytössä. Lisäksi hyökkääjän on pystyttävä tarkkailemaan uhrin liikennettä ja lähetettävä HTTP-pyyntöjä haavoittuvalle palvelimelle. Hyökkäys perustuu siihen, että hyökkääjä analysoi pakatun ja salatun vastauksen pituuden. HTTP-otsikoita ei pakata, joten esimerkiksi evästeitä ei voida tällä menetelmällä siepata. CSRF-avaimet (Cross-Site Request Forgery) ovat kuitenkin haavoittuvia. [81]

Jos hyökkääjä pyrkii murtamaan CSRF-avaimen, hänen on tiedettävä CSRF-avaimen muoto ja missä kohtaa se esiintyy HTTP-pyyntöissä. Oletetaan esimerkiksi, että CSRF-avain esitetään muodossa token=123456789. Hyökkääjä aloittaa arvaamalla ensimmäisen numeron ja lähettää palvelimelle pyyntöjä, joissa mukana on vain kyseinen arvaus. Kaikki mahdolliset numerot testataan, ja palvelimen lyhin vastaus osoittaa oikean arvauksen. Tätä prosessia toistetaan jokaisen numeron kohdalla, kunnes koko CSRF-avain on saatu selville. [81]

4.3.2 TLS

Kaikkien suojaamattomien protokollien suojaamiseen käytetään samaa TLS-salausprotokollaa, joten ne ovat kaikki alttiita TLS-protokollan tietoturvaongelmille. Vanhemmissa SSL- ja TLS-versioissa on useita tunnettuja haavoittuvuuksia. Alla on esitelty muutamia yleisesti tunnettuja hyökkäyksiä, jotka kohdistuvat näihin protokolleihin.

RC4-hyökkäys kohdistuu symmetriseen salaustekniikkaan. RC4 on symmetrinen virtasalausalgoritmi, joka julkistettiin vuonna 1987. RC4 otettiin käyttöön jo ensimmäisessä SSL 2.0 -versiossa, mutta se poistettiin käytöstä TLS 1.3 -versiossa. Toisin kuin nykyaikaisissa virtasalauksissa, RC4 ei käytä erillistä satunnaista alustusvektoria avaimen rinnalla. Tämä tarkoittaa, että jos yhtä pitkäaikaista avainta käytetään useiden datavirtojen salaamiseen, alustusvektori ja pitkäaikainen avain yhdistetään RC4:n salausavaimen luomiseksi.[82]

Ensimmäiset haavoittuvuudet RC4-algoritmissa löydettiin vuosina 2001 ja 2005. Nämä haavoittuvuudet mahdollistivat salausavaimen paljastamisen, jos suuri määrä viestejä salattiin samalla avaimella ja alustusvektori liitettiin yksinkertaisesti avaimen luomiseen. Huolimatta näistä tietoturvaongelmista, RC4:ään perustuvia SSL- ja TLS-salausohjelmistoja pidettiin yleisesti turvallisina aina vuoteen 2013 asti.[82]

Vuonna 2011 RC4:ää suositeltiin kiertotieksi BEAST-hyökkäystä vastaan. Kuitenkin vuonna 2013 julkaistut uudet hyökkäysmenetelmät osoittivat, että RC4 voidaan lopullisesti murtaa TLS:ssä. Hyökkäys vaatii noin 2^{34} salattua viestiä RC4-salauksen murtamiseen. Vuonna 2015 julkaistu uusi versio hyökkäyksestä alensi tämän vaatimuksen 2^{26} viestiin, tehden RC4:stä vielä haavoittuvamman.[82]

BEAST on selainhaavoittuvuus, joka julkaistiin vuonna 2011, ja se koskee SSL 3.0:aa ja TLS 1.0:aa. Hyökkääjä voi purkaa kahden osapuolen välillä vaihdetun tiedon salauksen hyödyntämällä haavoittuvuutta lohkosalausta käyttävän CBC (Cipher Block Chaining) -menetelmän toteutuksessa. BEAST on asiakaspuolen hyökkäys, joka on tyypiltään välimieshyökkäys. Hyökkääjä lisää paketteja TLS-virtaan ja tämän avulla pystyy arvaamaan viestiin käytetyn alustusvektorin. Näin hyökkääjä voi purkaa salauksen. Jotta BEAST-hyökkäys onnistuisi, hyökkääjän on saatava jollakin tavalla hallinta uhrin selaimessa. Nykyiset selainten versiot estävät BEAST-hyökkäyksen. RC4-virtasalaus on immuuni BEAST-hyökkäykselle, minkä vuoksi RC4:ää käytettiin laajalti palvelinpuolella BEAST-hyökkäyksen lieventämiseksi ennen kuin RC4 itsessään todettiin murretuksi vuonna 2013.[83]

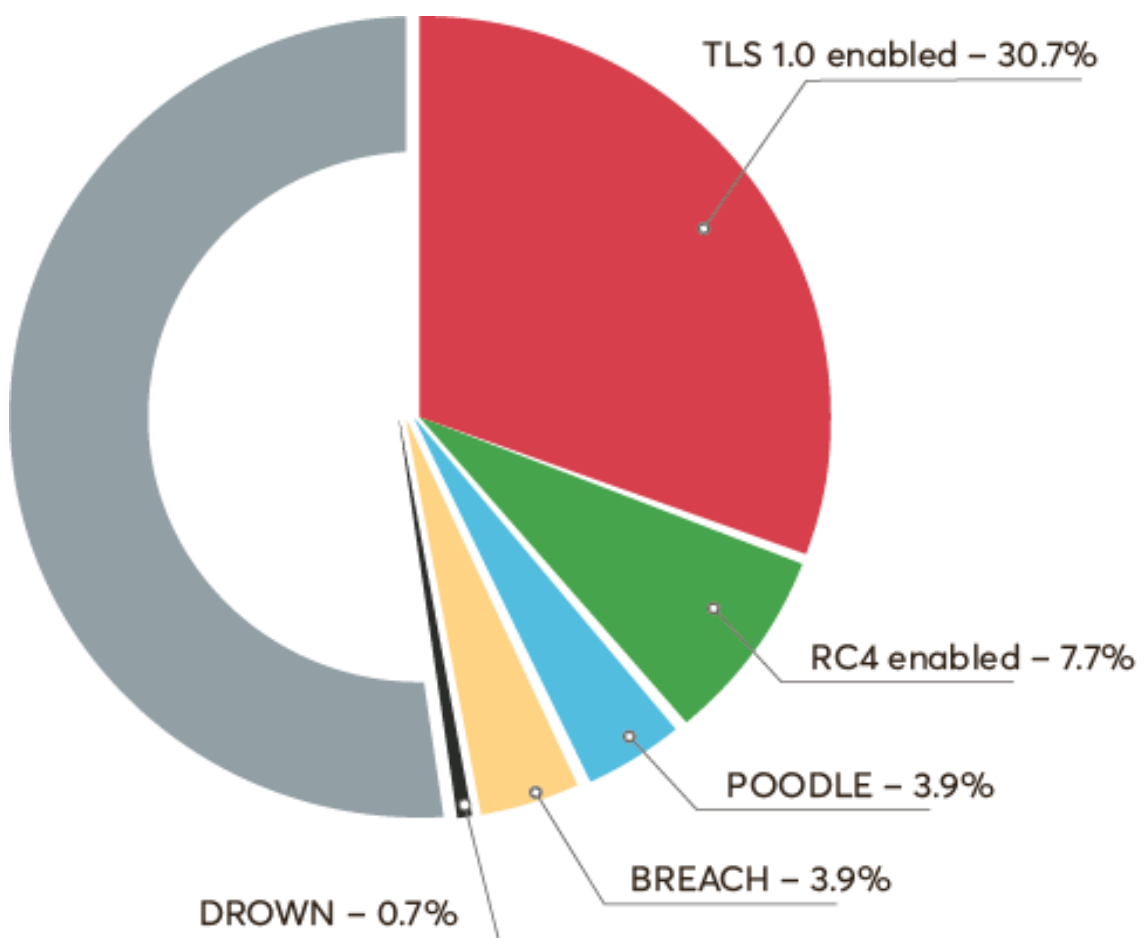
POODLE on SSL 3.0 -haavoittuvuus, joka johtuu, kuten BEAST-hyökkäyksessäkin, lohkosalauksen ketjuttamisen CBC (Cipher Block Chaining) -toteutuksesta. POODLE on välimieshyökkäys, ja se edellyttää, että hyökkääjän JavaScript-koodi suoritetaan uhrin selaimessa. POODLE-haavoittuvuus liittyy lohkosalausalgoritmien täytetarpeeseen: salattavan datan on oltava lohkokoon monikerta, tai muuten viimeinen lohko täytetään täytteellä. Jos data on täsmälleen lohkokoon monikerta, viimeinen lähetettävä lohko koostuu pelkästään täytteestä. Palvelin hylkää pyynnön, jos täyte on viallinen.[84][85]

POODLE-hyökkäys perustuu siihen, että hyökkääjä muokkaa pyynnön kokoa ja näin ollen tietää, minkälaista täytettä pyyntöön tulee. Täytteen sisältävä lohko on viimeisenä, ja hyökkääjä vaihtaa sen paikan murrettavan lohkon kanssa. Hyökkääjä lähettää muokatun pyynnön useita kertoja palvelimelle, muuttaen pyynnön kokoa jokaisella kerralla. Jos palvelin hyväksyy pyynnön, murrettavan lohkon bitit vastaavat täytelohkon täytteen bittejä. Hyödyntämällä tätä haavoittuvuutta hyökkääjä voi murtaa SSL-salauksen lohko kerrallaan. Esimerkiksi käyttäjän evästeiden varastaminen on onnistuneen POODLE-hyökkäyksen yksi sovellus. Esimerkiksi 16-bittisen evästeen varastamiseen tarvitaan maksimissaan 4096 pyyntöä, jos hyökkääjä tietää tarkalleen, missä kohdassa evästeet lähetetään pyynnössä.[84][85]

DROWN on SSL 2.0 -version haavoittuvuus, joka johtuu protokollan liian heikosta symmetrisestä salauksesta. Hyökkääjän on ensin kuunneltava satoja SSL-yhteyksiä uhrin ja palvelimen välillä. Yhteyksien täytyy jakaa jaettu salaus RSA-kättelyllä. Hyökkääjä lähettää palvelimelle muokattuja versioita uhrin salatuista RSA-kättelyistä. Tapa, jolla palvelin vastaa kuhunkin näistä muokatuista kättelyistä, riippuu siitä, onko salauksen purkamisen jälkeen saatu viesti oikean muotoinen. Koska salaus käyttää vain 40-bittistä avainta RSA-salauksessa, hyökkääjä pystyy vertaamaan palvelimen vastauksia kaikkiin mahdollisiin 2^{40} vaihtoehtoon. Näin hyökkääjä voi päätellä, oliko muokattu versio salatusta kättelystä oikein muodostettu vai ei.[86]

Tiedossa olevat TLS-haavoittuvuudet koskevat pääasiassa vanhentuneita TLS-versioita, joiden käyttöä ei enää suositella. RC4-salauksen haavoittuvuus koskee kuitenkin myös turvallisiksi katsottua TLS 1.2 -versiota. Kuva 37 esittelee Acunetix Web Application Vulnerability Report -tutkimuksen tuloksia vuodelta 2020. Kuva on otettu suoraan tutkimuksesta. Tutkimuksen on tehnyt tietoturvayhtiö Invicti, ja siinä on testattu 5000 palvelimen tietoturvaa. Tutkimuksen perusteella monet palvelimet sallivat edelleen TLS-yhteyden muodostamisen protokollan vanhoilla, haavoittuvilla versioilla.[87]

POODLE-hyökkäys edellyttää SSL 3.0 -versiota, ja DROWN-hyökkäys koskee vanhempaa SSL 2.0 -versiota. Silti 3,9 % testatuista palvelimista oli haavoittuvia POODLE-hyökkäykselle ja 0,7 % DROWN-hyökkäykselle. BREACH-hyökkäys johtuu HTTP-protokollan pakkausalgoritmista, ja 3,9 % palvelimista oli sille haavoittuvia. Lisäksi 7,7 % palvelimista salli käyttää murrettua RC4-algoritmia symmetrisenä salauksena. RC4-algoritmia voidaan käyttää kaikissa muissa TLS-versioissa paitsi TLS 1.3:ssa. Sen käyttö pitäisi erikseen kieltää TLS-palvelimen asetustiedostossa, jotta haavoittuvuudelta vältyttäisiin. Tutkimuksessa havaittiin myös, että 30,7 % palvelimista salli TLS 1.0 -version käytön, mikä tekee ne alttiiksi muun muassa BEAST-hyökkäykselle.

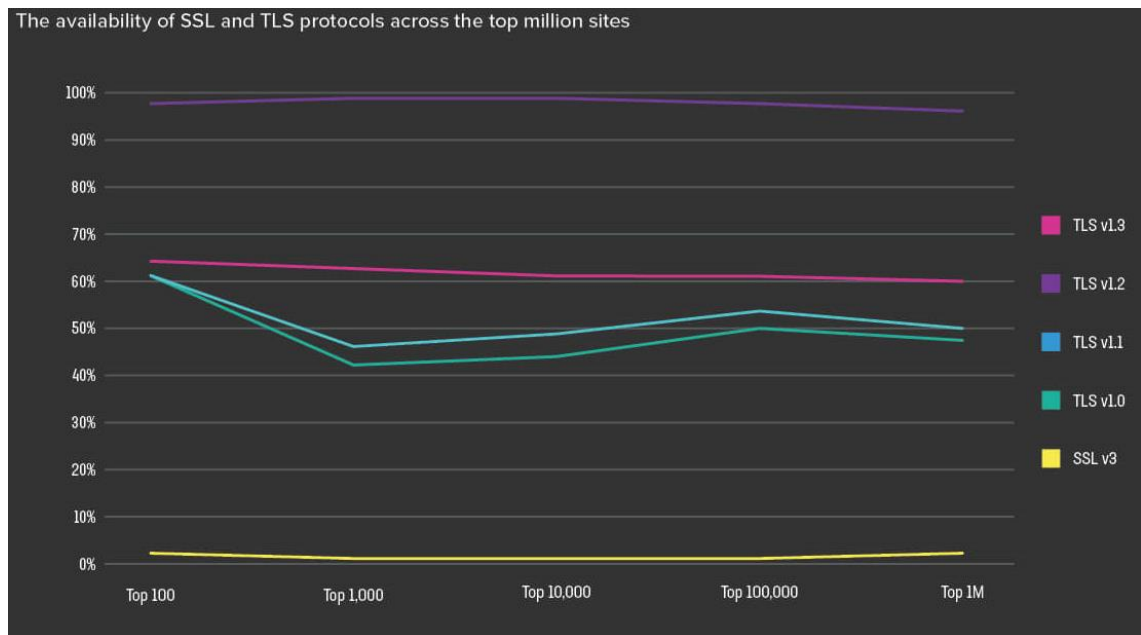


Kuva 37. Vuoden 2020 AWAVR- tutkimuksessa löydetyt TLS/SSL-haavoittuvuudet.[87]

Kuva 38 esittää F5 Labsin tekemän The 2021 TLS Telemetry Report -tutkimuksen tuloksia HTTPS-palvelimien tukemista SSL/TLS-versioista. Kuva on suoraan tutkimuksesta. Tutkimuksessa analysoitiin miljoona suosittua HTTPS-sivustoa. Tuloksista huomataan, että noin muutama prosentti tutkituista palvelimista mahdollistaa edelleen SSL 3.0 -version käytön. Nämä palvelimet ovat todennäköisesti haavoittuvia POODLE-hyökkäykselle.

TLS 1.0:aa tuki on noin 50 % palvelimista, mikä on huomattavasti suurempi luku kuin Acunetix Web Application Vulnerability Report -tutkimuksessa havaittu 30,9 %. TLS 1.1:llä on hieman enemmän tukea kuin TLS 1.0:lla. Ehdottomasti tuetuin versio on TLS 1.2, joka saavuttaa lähes 100 % tuen. Uusin versio, TLS 1.3, saavutti noin 60 % tuen tutkimuksessa. [88]

Vaikka F5 Labsin tekemä tutkimus ei suoraan paljasta tietoturva-avaavuuksia TLS-protokollassa, se antaa hyvän kuvan siitä, kuinka laajasti vanhentuneita, tietoturvaongelmista kärsiviä TLS-versioita käytetään. Näiden vanhojen versioiden käyttö mahdollistaa TLS-protokollaan kohdistettuja hyökkäyksiä. [88]



Kuva 38. Suosituimpien HTTPS-palvelimien tukemat TLS-versiot vuonna 2021 F5 Labsin tutkimuksessa.[88]

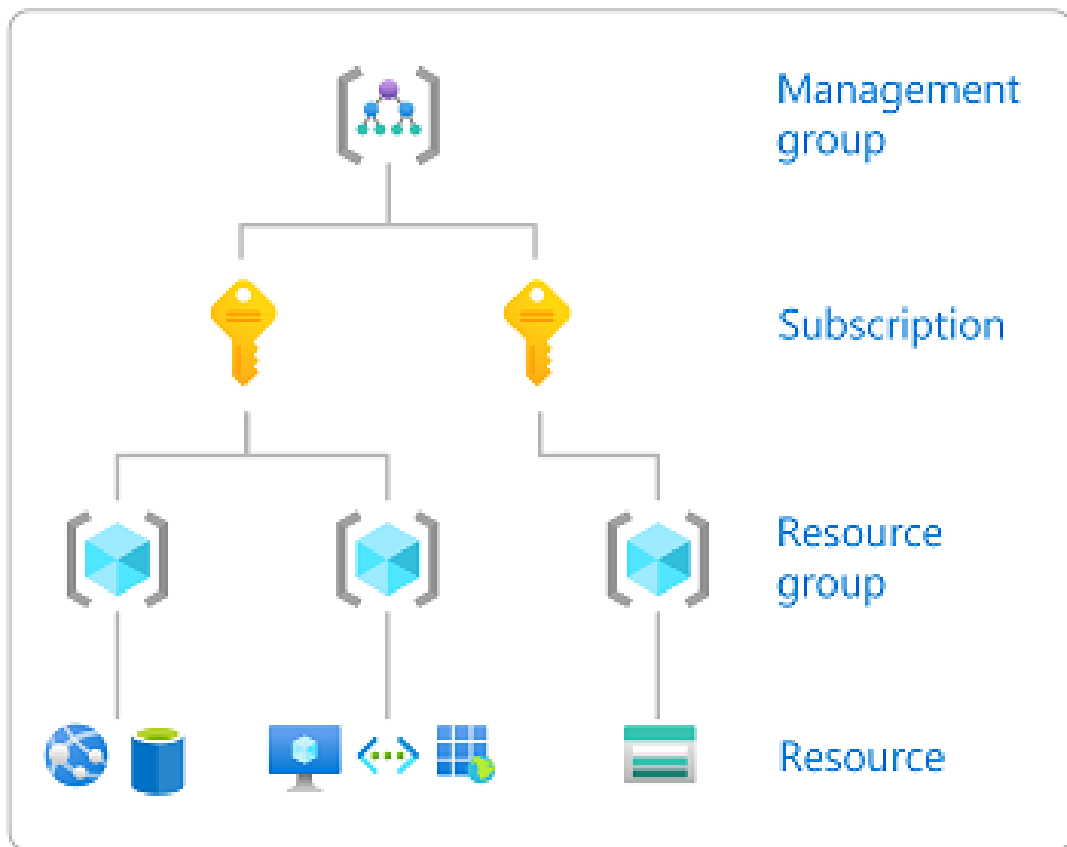
4.3.3 Suojatut protokollat

Kun FTPS- ja SFTP-protokollia vertaillaan tietoturvan näkökulmasta, oleellista on käytettävien porttien määrä. SFTP käyttää tiedonsiirtoon vain yhtä porttia, kun taas FTPS tarvitsee kaksi. Lisäksi jokainen rinnakkainen yhteys vaatii oman porttinsa. Tämän vuoksi FTPS-palvelimelle määritellään usein sallittujen porttien joukko palomuurin. Palomuuereja voidaan ajatella putkina, jotka säätelevät sallittua ja kiellettyä verkkoliikennettä yksityisessä verkossa. Ne luovat tarkastuspisteitä verkkoliikenteen ohjaamiseksi ja tarkistavat liikennettä ohjelmoitujen sääntöjen perusteella. Mitä vähemmän portteja on avattu, sitä vähemmän järjestelmässä on haavoittuvia kohtia. Jos käytössä on palomuuuri, se täytyy konfiguroida sallimaan kaikki FTPS-palvelimen käyttämät portit, mikä lisää tietoturvariskejä. [89]

FTPS-protokollan salaus perustuu TLS- tai SSL-tekniikoihin, jotka lisätään FTP-protokollan päälle. Tämän ulkoisen salauksen vuoksi palomuuriratkaisut eivät pysty havaitsemaan, mitä porttia käytetään ja miksi. Yhden portin SFTP-asetukset ovat ihanteellisia palomuurin rinnalla käytettäväksi, sillä ne muodostavat yhden yhteyden asiakkaan ja palvelimen välille. Palomuuri voi tarkkailla tämän yhteyden poikkeavuuksia, epäilyttäviä merkkejä ja muita uhkamerkkejä.[89]

4.3.4 Microsoft Azure

Azure käyttää roolipohjaista pääsynhallintaa (Azure RBAC) hallitakseen Azure-resurssien käyttöoikeuksia. Rooleja voidaan antaa käyttäjille, ryhmille, sovelluskäyttäjille tai hallituille identiteeteille, ja käyttöoikeudet resursseihin ovat sidottuja rooleihin. Azure tarjoaa valmiiksi määriteltäviä rooleja, ja käyttäjät voivat myös luoda omia mukautettuja rooleja. Rooleille on määriteltävä laajuus, joka tarkoittaa joukkoa resursseja, joita käyttöoikeus koskee. Azuressa laajuus voidaan määrittää neljällä tasolla, laajimmasta kapeimpaan: hallintaryhmä, tilaus, resurssiryhmä ja resurssi. Kuva 39 esittää Azuren laajuuksien hierarkian.[90][91]



Kuva 39. Azuren roolien laajuus.[90]

Hierarkian alimpana tasona on yksittäinen resurssi, joka tarkoittaa jotakin Azuren tarjoamaa palvelua. Sen yläpuolella on resurssiryhmä, jonka avulla resursseja voidaan ryhmitellä. Jokainen resurssi kuuluu yhteen resurssiryhmään, ja ryhmässä voi olla useita resursseja. Resurssiryhmän yläpuolella on tilaus, johon voi kuulua useita resurssiryhmiä. Kun käyttäjätili luodaan Azureen, siihen sisältyy yksi tilaus. Tilaus tarjoaa laskutuksen sekä statistiikkaa käytetyistä resursseista ja niiden kustannuksista. Käyttäjätiliin voidaan lisätä lisää tilauksia, joista jokaiselle muodostetaan oma lasku. Useita tilauksia voidaan hallita yhdessä hallintaryhmän avulla.[90][91][92]

Kaikki käyttäjät, joilla on käyttöoikeus Azuren resursseihin, voivat käyttää niitä omilla Microsoft-tunnuksillaan joko selainpohjaisen Azure Portalin kautta tai CLI-työkalun avulla. Molemmat työkalut edellyttävät käyttäjänimen ja salasanan syöttämistä. Sovelluskehityksessä käyttäjänimen ja salasanan käyttö ei kuitenkaan ole suositeltavaa, sillä oman salasanan suojaaminen muilta kehittäjiltä tai ylläpitäjiltä on haastavaa. Lisäksi käyttäjillä on usein laajemmat käyttöoikeudet kuin kehitettävä sovellus tarvitsisi. [93][94][95][96]

Tätä varten pääsynhallinnassa käytetään sovelluskäyttäjiä ja hallittuja identiteettejä. Sovelluskäyttäjät liittyvät Microsoftin laajempaan pääsynhallintajärjestelmään, Microsoft Entra ID:tä. Entra ID kattaa myös muita Microsoftin palveluita, kuten Office 365 ja OneDrive, ja sen kautta voidaan hallita pääsyoikeuksia näihin palveluihin. Entra ID:llä on oma selainpohjainen hallintapaneeli, jonka avulla pääsyoikeuksia voidaan hallita. Sovelluskäyttäjille voidaan antaa käyttöoikeuksia eri Azuren resursseihin Azure-roolien avulla. [93][94][95][96]

Hallitut identiteetit ovat sovelluskäyttäjien erikoistapauksia ja toimivat samalla tavalla. Ainoa ero on, että hallitut identiteetit toimivat vain Azuren sisällä, ja niiden käyttöoikeudet rajoittuvat Azuren resursseihin. Sovelluskäyttäjät ja hallitut identiteetit mahdollistavat kirjautumisen sertifikaatin tai pitkän satunnaissgeneroidun salasanan avulla. Niille voidaan myöntää juuri ne käyttöoikeudet, joita sovellus tarvitsee, ja käyttöoikeuksia voidaan muokata tarvittaessa. [93][94][95][96]

Azure tarjoaa mahdollisuuden hallita yksittäisen tallennusratkaisun pääsyä 512-bittisten käyttöavaimien avulla. Käyttöavaimia on kaksi, jotka mahdollistavat sekä kirjoitus- että lukuoperaatiot, ja kaksi avainta, jotka mahdollistavat pelkät lukuoperaatiot. Käyttöavaimet ovat käytössä seuraavissa tallennusratkaisuissa: Cosmos-tietokanta, File Share ja Blob Storage. Käyttöavaimen avulla tallennusratkaisua voidaan käyttää sen julkisen päätepisteen tai ohjelmistokielen SDK-työkalujen avulla. Molemmat lukuoikeuden antavat avaimet tarjoavat samanlaiset käyttöoikeudet

tallennusratkaisuun, ja samalla tavalla kirjoitus- ja lukuoikeuden antavat avaimet tarjoavat yhtäläiset oikeudet. Kahden avaimen olemassaolo mahdollistaa avainten kierrättämisen. Tämä tarkoittaa, että toinen avain voidaan päivittää uuteen avaimeseen ilman palvelun käyttökatkosta, koska sovellus voi edelleen käyttää toista, vanhempaa avainta. Kierrättämisellä pyritään parantamaan tietoturvaa, ja Microsoft suosittelee avainten säännöllistä kierrättämistä.[97][98][99]

Käyttöavainten avulla voidaan luoda jaettuja käyttöoikeusallekirjoituksia (SAS). Jaettu käyttöoikeusallekirjoitus on URL, joka myöntää rajoitetut käyttöoikeudet tallennusratkaisuun. Jaetulle käyttöoikeusallekirjoitukselle voidaan määrittää, mitä operaatioita sen avulla voidaan suorittaa sekä allekirjoituksen voimassaoloaika. Jaettu käyttöoikeusallekirjoitus on hyödyllinen tilanteissa, joissa halutaan antaa pääsy vain tiettyihin tallennustilin resursseihin tietyksi ajaksi.[97][98][99]

Microsoft Entra ID:n aikaisempi nimi oli Azure Active Directory (AAD), ja siitä löydettiin vakava haavoittuvuus vuonna 2023. Wiz-tutkimustiimi löysi haavoittuvuuden Microsoftin omistamasta Bing-hakukoneesta, jonka pääsynhallinta oli toteutettu AAD:n avulla. Bing-hakukoneen pääsynhallinnassa oli haavoittuvuus, joka mahdollisti minkä tahansa AAD-käyttäjän kirjautumisen Bing Trivia -sivustolle. Bing Trivia oli hallintapaneeli, jonka avulla voitiin muun muassa muokata Bing-hakukoneen hakusivun ulkoasua. Wiz-tutkimustiimi onnistui lisäämään Bing-hakukoneeseen haitallista JavaScript-koodia. Tätä hyökkäystä kutsutaan Cross Site Scripting (XSS) -hyökkäykseksi, jossa hyökkääjä onnistuu ajamaan haitallista JavaScript-koodia uhrin selaimessa. XSS-hyökkäyksen avulla pystyttiin varastamaan Bing-käyttäjien Office 365 -tileihin kirjautumistunnuksia. Lisäksi muihin pienempiin Microsoftin sisäisiin palveluihin pääsi kirjautumaan millä tahansa AAD-tunnuksella.[100]

Azuren tallennusratkaisuista löydettiin vakava tietoturva haavoittuvuus vuonna 2021, joka sai nimekseen ChaosDB. Haavoittuvuus koski Cosmos-tietokantaa, ja sen löysi Wiz-tutkimustiimi. ChaosDB-haavoittuvuus mahdollisti rajoittamattoman pääsyn tuhansien Azure-asiakkaiden Cosmos-tietokantoihin. Haavoittuvuus johtui Jupyter Notebook -ominaisuuden virheellisestä konfiguraatiosta Cosmos-tietokantaan. Jupyter Notebook on interaktiivinen verkkosovellus, jota käytetään laskennallisten asiakirjojen luomiseen ja jakamiseen. Se mahdollistaa Cosmos-tietokannan datan visualisoinnin. Hyödyntämällä Jupyter Notebookin haavoittuvuuksien ketjua hyökkääjä pystyi tiedustelemaan tietoja kohde-Cosmos-tietokannan Jupyter Notebookista. Tämän seurauksena hyökkääjä sai käsiinsä tunnistetietoja, jotka liittyivät kohde-Cosmos-tiliin sekä Jupyter Notebook -laskentaan. Näihin tunnistetietoihin kuului myös Cosmos-

tietokannan käyttöavaimet, joiden avulla hyökkääjä pääsi käsiksi tietokantaan.[101][102]

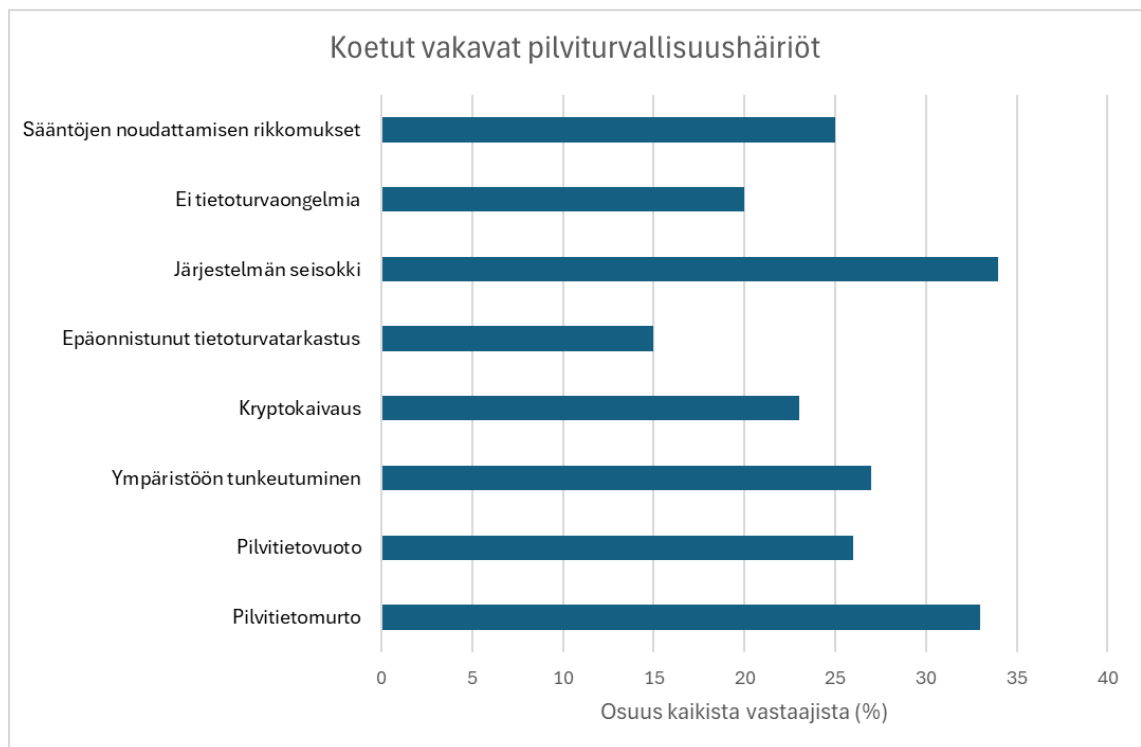
ChaosDB-haavoittuvuus mahdollisti hyökkääjien nostaa käyttöoikeuksiaan, mikä avasi pääsyn Cosmos-tietokannan sisäisiin salaisuuksiin ja varmenteisiin. Näiden salaisuuksien avulla hyökkääjät pystyivät pääsemään muiden Cosmos-tietokantojen käyttöavaimiin. Haavoittuvuus koski kaikkia Cosmos-tietokantoja, joissa Jupyter Notebook -ominaisuus oli kytketty päälle. Ominaisuus oli oletuksena päällä uusissa Cosmos-tietokannoissa, ja käyttäjän täytyi erikseen poistaa se käytöstä tietokantaa luodessa. Wiz-tutkimustiimi ilmoitti haavoittuvuudesta Microsoftille, joka korjasi ChaosDB-haavoittuvuuden kahden päivän kuluessa. Microsoft ilmoitti, ettei ole viitteitä siitä, että mikään muu taho kuin Wiz-tutkimustiimi olisi hyödyntänyt haavoittuvuutta ja päässyt muiden käyttäjien Cosmos-tietokantoihin. Microsoft kuitenkin suositteli kaikkia asiakkaitaan kierrättämään Cosmos-tietokannan käyttöavaimet. [101][102]

Azure tarjoaa osalle tallennusratkaisuista lisäpalveluna erillisen tietoturvapalvelun. Microsoft Defender for Storage on Azuren tietoturvakeros, joka havaitsee mahdolliset uhat tallennusratkaisuissa. Se auttaa estämään vakavia tietoturvaongelmia, kuten haitalliset tiedostojen lataukset, arkaluonteisten tietojen vuotamisen ja tietojen korruption. Palvelu on käytettävissä Blob Storage-, Azure Files- ja Azure Data Lake Storage -tallennusratkaisuiden kanssa. [103]

Microsoft Defender for Storage tarjoaa seuraavat toiminnallisuudet: toiminnan seuranta, arkaluonteisten tietojen uhkien havaitseminen ja haittaohjelmien tarkistus. Toiminnan seurannassa Defender for Storage analysoi jatkuvasti suojattujen tallennusratkaisuiden dataa ja ohjaustason lokeja, kun se on käytössä. Palvelu käyttää Microsoft Threat Intelligence -järjestelmää tunnistamaan epäilyttävät toiminnot, kuten haitalliset IP-osoitteet, Tor-poistumissolmut ja mahdollisesti vaaralliset sovellukset. Lisäksi se rakentaa tietomalleja ja hyödyntää tilastollisia sekä koneoppimismenetelmiä havaitakseen perustoiminnasta poikkeavat tapahtumat, jotka voivat viitata haitalliseen toimintaan. Palvelu luo tietoturvaravitteitä epäilyttävistä toiminnoista. Haittaohjelmien tarkistus auttaa suojaamaan tallennustilit haitalliselta sisällöltä suorittamalla täyden haittaohjelmatarvituksen ladatulle sisällölle lähes reaaliajassa käyttäen Microsoft Defender Antivirus -ominaisuuksia. Arkaluonteisten tietojen uhkien havaitseminen mahdollistaa tietoturvaravitteiden priorisoinnin tietojen arkaluonteisuuden perusteella. Defender for Storage maksaa 10 € kuukaudessa tallennustiliä kohden.[103]

4.3.5 Pilvipalveluiden sovellusten tietoturvatutkimukset

Vaikka pilvipalveluiden tietoturvaongelmat ovat erittäin harvinaisia, pilvipalveluihin kehitettyjen sovellusten tietoturvaongelmat ovat yleisiä. Snyk-tietoturvayhtiön vuonna 2022 tekemässä The State of Cloud Security Report 2022 -tutkimuksessa tutkittiin pilvipalveluihin kehitettyjen sovellusten tietoturvahaasteita. Tutkimukseen osallistui tietoturva-asiantuntijoita ja pilvikehittäjiä. Kuva 40 esittää tutkimukseen osallistuneiden raportoidut vakavat tietoturvaongelmat pilvipalveluihin kehitetyissä sovelluksissa. Kaikki tässä työssä esitetyt kuvat The State of Cloud Security Report 2022 -tutkimuksesta on luotu Excel-ohjelmalla, käyttäen tutkimuksessa julkaistuja tuloksia. Yleisin ongelma oli järjestelmän seisokki, joka oli johtunut virheellisestä konfiguroinnista. Tietomurrot ja tietovuodot olivat myös erittäin yleisiä: vastaajista noin 26 % oli kokenut tietovuodon ja 33 % tietomurron.[104]

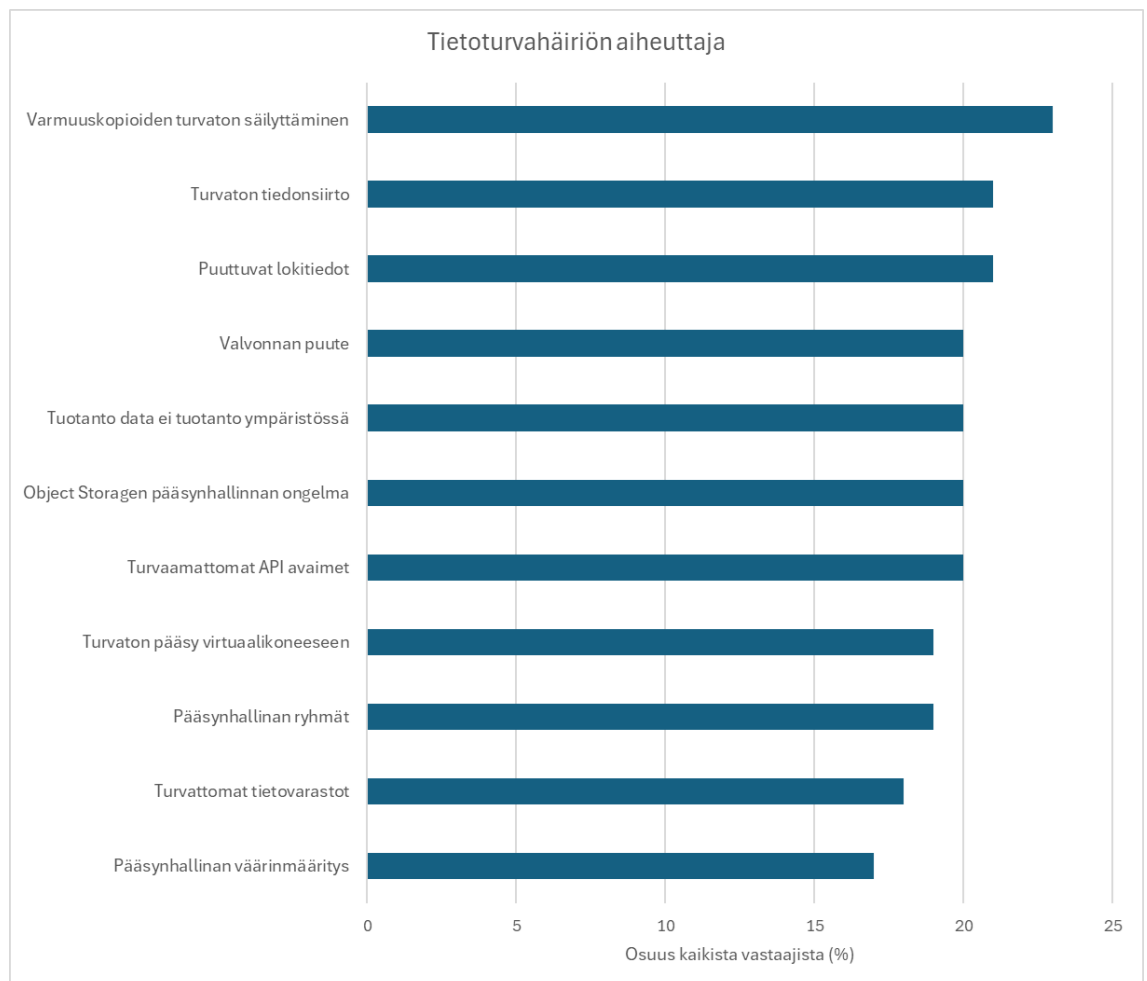


Kuva 40. Snykin tutkimukseen vastanneiden kokemat pilviturvallisuusongelmat. [104]

Samassa tutkimuksessa kysyttiin myös tietoturvaongelmien aiheuttajia. Vastaajat saivat valita useita teknisiä syitä, jotka olivat aiheuttaneet tietoturvaongelmia pilvipalveluissa. Kuva 41 esittää tutkimuksen tuloksia teknisistä ongelmista, jotka olivat johtaneet tietoturvahäiriöihin. Tutkimuksen mukaan eniten tietoturvaongelmia aiheuttivat turvattomasti säilytetyt varmuuskopiot, jotka olivat osallisina 23 % tapahtuneista tietoturvaongelmista. [104]

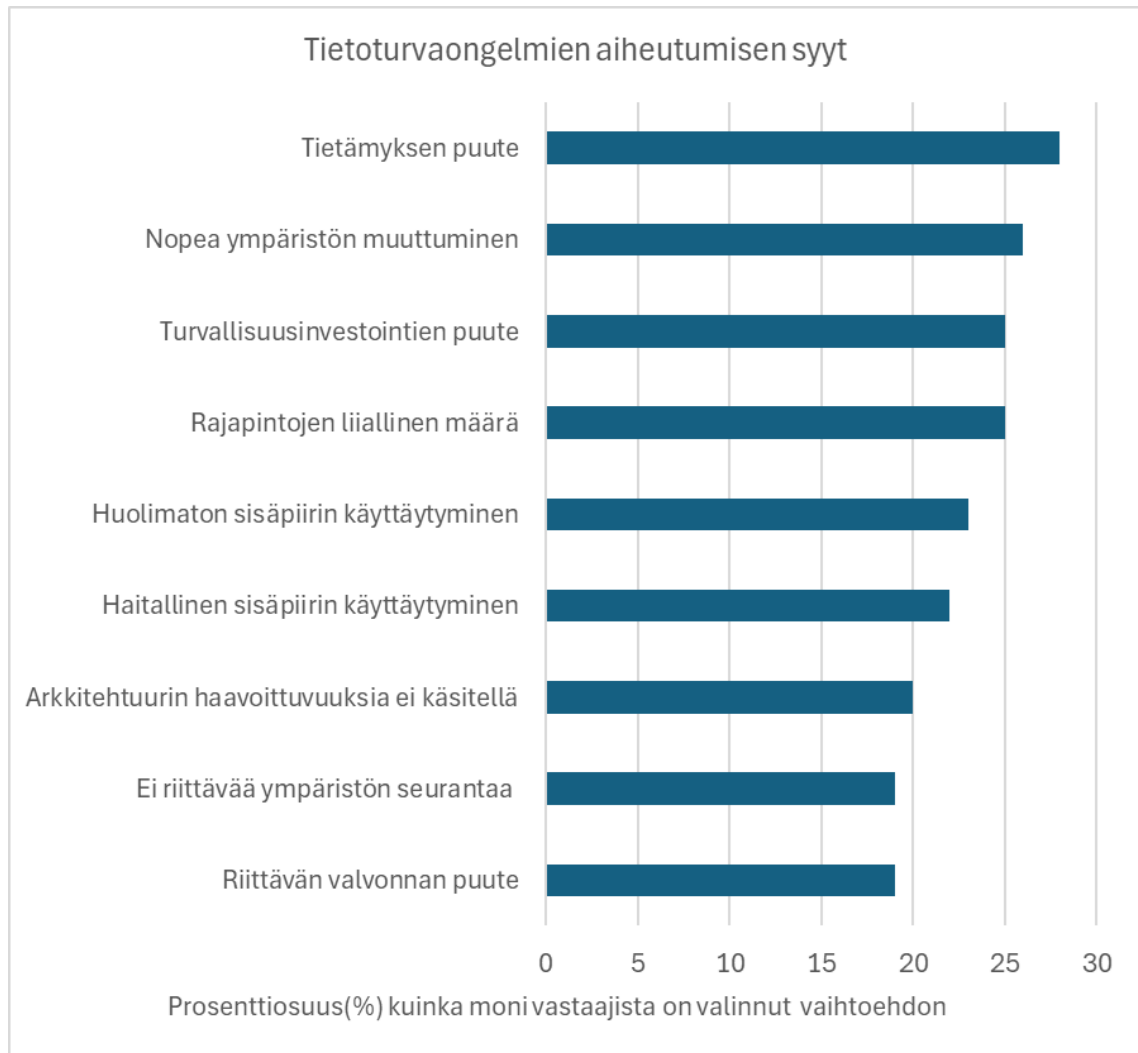
Ongelman aiheuttajat voidaan ryhmitellä muutamaaan pääryhmään:

- Tietoturvakriittisten asioiden, kuten varmuuskopioiden ja API-avainten, turvaton säilyttäminen.
- Puutteelliset valvontamenetelmät, kuten riittämättömät lokitiedot.
- Pääsynhallinnan virheelliset asetukset, sisältäen käyttäjäryhmät ja Object storagen.
- Turvattomat yhteydet ja tietovarastot, kuten virtuaalikoneiden yhteydet, tiedonsiirto ja tietovarastot.



Kuva 41. Tietoturvaongelmien aiheuttajat Snykin tutkimuksessa. [104]

Samassa tutkimuksessa kysyttiin myös syitä, jotka olivat aiheuttaneet tietoturvaongelmia. Kuva 42 esittää tutkimuksen mukaan yleisimmät syyt tietoturvaongelmien taustalla.

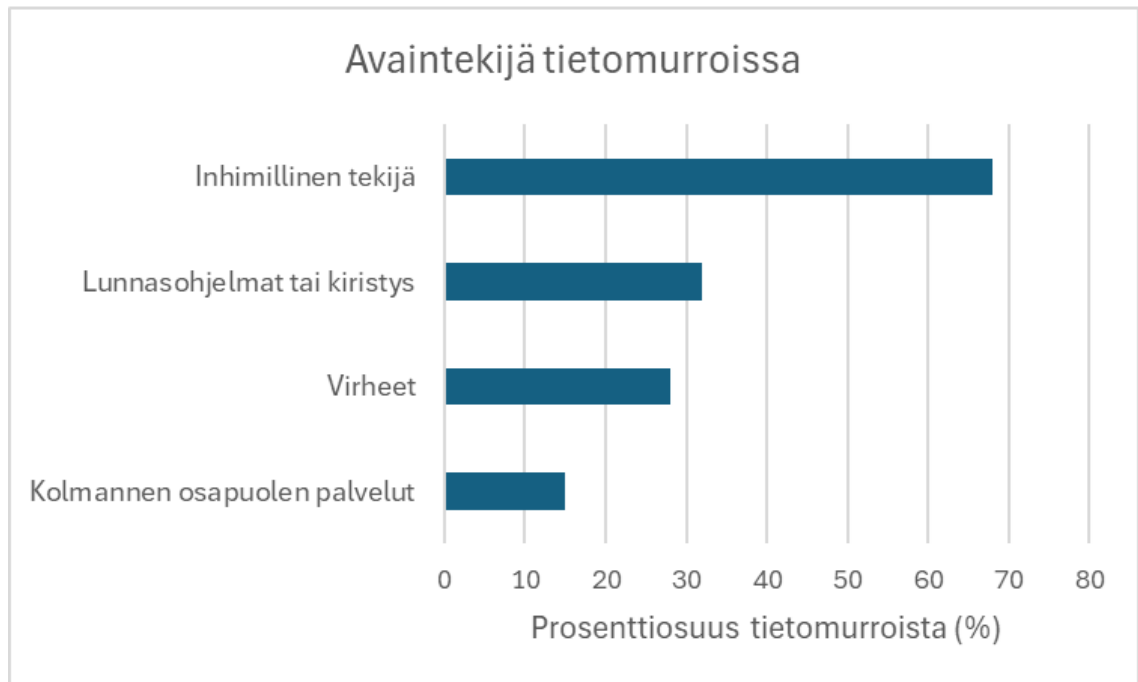


Kuva 42. Tietoturvaongelmien aiheutumisen syyt Snykin tutkimuksen mukaan. [104]

Tietämyksen puute tietoturvasta ja nopea ympäristön muuttuminen olivat yleisimmät syyt, jotka johtivat tietoturvaongelmiin. Tutkimuksen mukaan nämä syyt todennäköisesti lisääntyvät tulevaisuudessa, kun yhä useammat tiimit ottavat käyttöön jatkuvan käyttöönotton menetelmän, mikä tuo mukanaan uusia tietoturva haasteita. Suurin osa syistä johtui ihmisten toiminnasta, eikä ollut suoraan yhteydessä käytettyyn pilviteknologiaan. [104]

Inhimillisen tekijän vaikutusta tietoturvaongelmiin on tutkittu IBM:n vuonna 2024 julkaisemassa *2024 Data Breach Investigations Report* -tutkimuksessa. IBM:n tutkimus käsittelee vuosina 2022–2023 tapahtuneita tietomurtoja, mukaan lukien pilvipalveluiden ulkopuoliset tietomurrot. Kaikki tässä työssä esitetyt kuvat *2024 Data Breach Investigations Report* -tutkimuksesta on luotu Excel-ohjelmalla, käyttäen tutkimuksessa julkaistuja tuloksia. Kuva 43 esittää IBM:n tutkimuksen avaintekijät tietomurroissa. Inhimillinen tekijä oli yleisin ja mukana 68 % tietomurroista. Muut tutkimuksessa

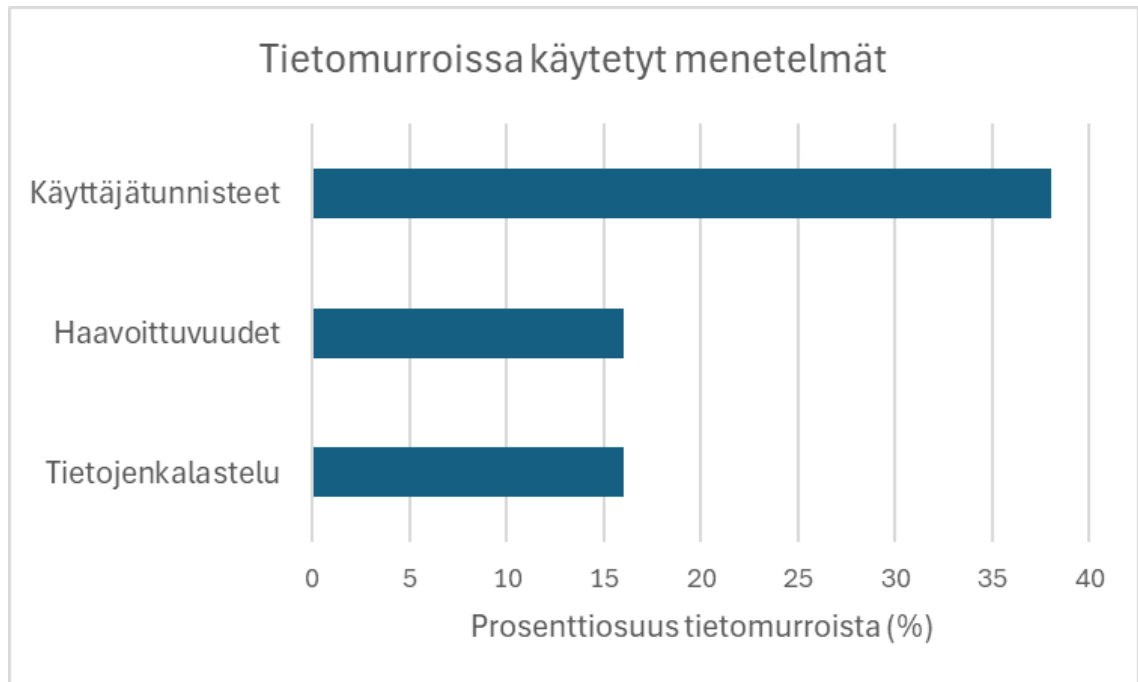
mainitut tekijät olivat teknisiä, mutta niiden yleisyys oli korkeintaan puolet inhimillisen tekijän osuudesta.[105]



Kuva 43. Avaintekijät tietomurroissa IBM:n tekemässä tutkimuksessa. [105]

Yksi IBM:n tutkimuksen inhimillisistä tekijöistä on tietojenkalastelu. Tietojenkalastelu tarkoittaa yritystä varastaa arkaluonteisia tietoja, kuten käyttäjätunnuksia, salasanoja tai pankkitunnuksia. Tietojenkalastelu toteutetaan yleensä sähköpostin tai tekstiviestien avulla. Vuonna 2023 tehdyssä tutkimuksessa 20 % vastaajista ilmoitti saaneensa tietojenkalasteluviestejä, ja 11 % oli avannut viestin. Tietojenkalastelun määrä on kasvanut tasaisesti vuodesta 2016 lähtien, jolloin tutkimuksessa alettiin ensimmäistä kertaa seurata tietojenkalasteluviestejä.[105]

Kuva 44 esittää IBM:n tutkimuksessa käytettyjä menetelmiä tietomurroissa. Yleisin menetelmä oli käyttäjätunnukset, joka sisältää virheellisen pääsynhallinnan ja turvattoman tavan säilyttää käyttäjätunnuksia. Tietojenkalastelu oli tutkimuksen mukaan toiseksi yleisin tietomurtojen aiheuttaja yhdessä haavoittuvuuksien hyväksikäytön kanssa.[105]



Kuva 44. Tietomurroissa käytetyt menetelmät IBM:n tutkimuksessa. [105]

5. YHTEENVETO

Työssä tutkittiin tiedostojen siirtoa ja tallennusta Azure-pilvipalvelussa kustannusten, tehokkuuden ja tietoturvan näkökulmasta. Tiedostojen siirtoa analysoitiin eri tiedonsiirtoprotokollien avulla, ja tallennusta vertailtiin Azuren tarjoamien tallennusratkaisujen kesken. Työtä varten luotiin Azureen testausympäristö, jossa suoritettiin tarvittavat testit. Testausympäristöön pystytettiin virtuaalikoneeseen tiedonsiirtoprotokollien edellyttämät palvelimet, jotka yhdistettiin tallennusratkaisuihin. Testit suoritettiin kannettavalla tietokoneella käyttäen Python- ja C#-ohjelmointikieliä sekä selainta.

Tallennuksen kustannusvertailussa keskityttiin Azuren tallennusratkaisujen hintoihin. Tallennusratkaisuiden hintojen komponentit esiteltiin ja hinnoista muodostettiin taulukot eri redundanssiasetuksilla. Kalleimpia olivat hallinnoidut tietokantaratkaisut SQL- ja Cosmos-tietokanta. Molemmat tarjoavat täysin Azuren ylläpitämän eheyden hallinnan tiedostoille sekä varmuuskopioinnin. Useat käyttäjät pystyvät muokkaamaan tiedostoja yhtäaikaaisesti sekä tekemään kyselyjä tietokantaan. Muut tallennusratkaisut eivät mahdollista monimutkaisten kyselyjen tekimiseen, vaan ne tarjoavat yksinkertaisen rajapinnan tiedostojen tallentamiseen. Tietokantojen tallennuskustannukset olivat monta kertaa kalliimpia kuin muiden tallennusratkaisuiden kalliimmat palvelumallit.

Kaikki muut tallennusratkaisut, paitsi tietokannat, tarjoavat kylmiä ja kuumia palvelumalleja. Kuumat palvelumallit ovat kalliimpia tallennustilan osalta, mutta edullisempia ja nopeampia I/O-operaatioiden suhteen, tarjoten parhaan suorituskyvyn. Kylmät palvelumallit on taas tarkoitettu pitkäaikaiseen säilytykseen, ja niiden tallennustila on edullista, mutta I/O-operaatiot ovat kalliita ja hitaita. Näiden ratkaisujen avulla voidaan tarjota joko korkeaa suorituskykyä sitä vaativille käyttäjille tai edullisia pitkäaikaisia säilytysratkaisuja.

Blob Storagen arkisto-palvelumalli erottui ehdottomasti edullisimmaksi tallennusratkaisuksi tallennustilan osalta, mutta I/O-operaatiot ovat kaikista kalleimmat. Lisäksi arkisto-palvelumallissa tiedoston lataaminen voi pahimmillaan kestää useita tunteja, vaikka tiedoston tallentaminen tapahtuu yhtä nopeasti kuin muissa Blob Storagen palvelumalleissa. Arkisto-palvelumalli sopii nimensä mukaisesti parhaiten varmuuskopioiden sekä muiden kopioiden ja lokien pitkäaikaiseen säilytykseen.

Kun vertailtiin kuumien ja kylmien palvelumallien suorituskykyä, havaittiin, että kaikista kylmimmän palvelumallin suorituskyky oli selvästi heikompi. Sen sijaan muiden, kuumempien palvelumallien välillä ei ollut merkittäviä suorituskykyeroja. Tässä työssä tallennusratkaisuille ei aiheutettu IO-kuormaa testien aikana, ja tulokset voisivatkin olla erilaiset suurella IO-kuormalla, sillä kuumimmat palvelumallit tarjoavat parhaat IOPS-suorituskyvyt.

Kuumien palvelumallien välillä havaittiin kuitenkin suuria kustannuseroja, joten käyttäjän täytyy tarkoin harkita valitsemaansa palvelumallia. On olemassa riski, että käyttäjä maksaa turhaan kalliista kuumasta palvelumallista, joka tarjoaa saman suorituskyvyn kuin edullisempi kylmä palvelumalli. Myös muissa tutkimuksissa on saatu vastaavia tuloksia AWS S3 -palvelun kuuman palvelumallin heikosta suorituskyvystä.

Tallennusratkaisujen redundanssiasetuksella on suuri vaikutus tallennusratkaisun hintaan. Mitä laajemmalle alueelle kopiot levitetään, sitä kalliimmaksi tallennusratkaisu muuttuvat. Mitä laajemalla alueella kopiot levitetään sitä vähemmän tallennusratkaisulla on tarjolla eri palvelumalleja. Laajempi levitys takaa paremman häiriönsietokyvyn, ja tallennusratkaisut ovat käytettävissä suuremman osan ajasta. Ilmoitetut käytettävyyssajat ovat erittäin suuria: halvimman vaihtoehdon käyttökatko vuodessa on vain noin puolikkaan millisekunnin, ja kalliimpien ratkaisujen käyttökatko on siitä vain noin miljoonasosan. Redundanssiasetuksella ei ollut vaikutusta tallennusratkaisun suorituskykyyn.

Tiedostojen siirron kustannuksia vertailtiin epäsuorasti siirretyn internetliikenteen määrän perusteella. Protokollien välillä havaittu tiedostokoon kasvu oli vähäistä, eikä sillä ollut merkittävää vaikutusta siirtokustannuksiin. SFTP-protokolla kasvatti eniten siirrettävän tiedoston kokoa, noin 4,7 % suuremmalla 200 MB tiedostolla ja 6,7 % pienemmällä 1 MB tiedostolla.

Tiedostojen siirron kustannuksilla ei ole käytännössä merkittävää vaikutusta, kun niitä verrataan tallennusratkaisujen tai muiden pilvipalveluiden kustannuksiin. Vaikka osa pilvipalveluista laskuttaa verkkoliikenteestä, muutaman prosentin kasvu siirtokustannuksissa ei todennäköisesti vaikuta kokonaiskustannuksiin merkittävästi. Lisäksi tällainen pieni kasvu verkkoliikenteessä ei aiheuta lisäkuluja julkisen internetliikenteen vuoksi.

Tallennuksen tehokkuusvertailussa analysoitiin tiedostojen tallennusnopeutta eri tallennusratkaisuihin. Sisäverkon ratkaisut osoittautuivat tehokkaimmiksi, kun taas julkisen HTTP-rajapinnan kautta toimivat ratkaisut olivat hitaimpia. Cosmos-tietokanta, joka käyttää JSON-dokumentteja, ei toiminut lainkaan binääristen tiedostojen

tallennuksessa. Kaikista parhaimman suorituskyvyn tarjosi Managed Disks -palvelu, jonka tarjoamalla SSD-kovalevyllä oli kaikista paras suorituskyky.

Tiedostojen siirron tehokkuusvertailussa tutkittiin eri tiedonsiirtoprotokollien suorituskykyä. Salatut protokollat olivat keskimäärin hieman hitaampia kuin salaamattomat, mutta ero suorituskyvyssä oli vähäinen. Työssä paras suorituskyky saavutettiin WebSocket-protokollalla, vaikka sitä ei ole erityisesti optimoitu tiedostojen siirtoon, vaan sitä käytetään pääasiassa reaaliaikaisten muutosten päivittämiseen internetsivuilla. Tulos on todennäköisesti virheellinen juuri WebSocketin käyttötarkoituksen vuoksi. Sitä käytetään pääasiassa reaaliaikasovelluksissa, ei sovelluksissa, joissa siirretään suuria määriä dataa. Eräässä toisessa tutkimuksessa havaittiin, että WebSocketin suorituskyky oli huomattavasti parempi kuin HTTP-protokollan, kun palvelimelle lähetettiin tuhansia kopioita pienestä tekstitiedostosta. Tämä tulos on todennäköisesti perusteltu WebSocketin käyttötarkoituksen vuoksi, sillä se on suunniteltu siirtämään lyhyitä viestejä reaaliajassa.

Tietoturva-vertailussa esiteltiin suojaamattomien tiedonsiirtoprotokollien tietoturvaongelmat sekä suojattujen tiedonsiirtoprotokollien vaatimat palomuuriasetukset. TLS-salausta käytetään suojaamaan alun perin suojaamattomat tiedonsiirtoprotokollat, ja siitä on olemassa useita vanhentuneita versioita, joita monet palvelimet edelleen tukevat. Tutkimuksissa on todettu, että nykypäivän palvelimet ovat haavoittuvia hyvin tunnetuille TLS-haavoittuvuuksille, jotka ovat olleet olemassa yli kymmenen vuotta. Myös Azure-palveluista on löydetty muutamia vakavia tietoturva-aukkoja, mutta yleisesti voidaan todeta, että pilvipalvelut ovat erittäin turvallisia käyttää. Sen sijaan pilvipalveluihin kehitettyjen palveluiden tietoturvassa on havaittu useita ongelmia. Suurin syy tietoturvaongelmiin on ollut inhimilliset virheet ja riittämätön tietämys pilvipalveluista sekä tietoturvasta. Lisäksi haavoittuvuuksien kautta on onnistuttu tekemään tietomurtoja.

LÄHTEET

- [1] <https://www.dataversity.net/brief-history-cloud-computing/>, viitattu 3.9.2024
- [2] <https://www.dataversity.net/how-the-cloud-has-evolved-over-the-past-10-years/>, viitattu 3.9.2024
- [3] <https://www.cloudzero.com/blog/cloud-computing-statistics/>, viitattu 3.9.2024
- [4] <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-are-private-public-hybrid-clouds>, viitattu 3.9.2024
- [5] <https://www.marketsandmarkets.com/Market-Reports/cloud-storage-market-902.html>, viitattu 3.9.2024
- [6] <https://www.cloudflare.com/learning/ddos/glossary/hypertext-transfer-protocol-http/>, viitattu 4.9.2024
- [7] <https://httpwg.org/specs/rfc9110.html>, viitattu 4.9.2024
- [8] <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview>, viitattu 4.9.2024
- [9] <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>, viitattu 4.9.2024
- [10] <https://www.javacodegeeks.com/2023/03/http-1-1-vs-http-2-vs-http-3-key-differences.html>, viitattu 4.9.2024
- [11] <https://ably.com/topic/http-2-vs-http-3>, viitattu 4.9.2024
- [12] <https://www.w3.org/Protocols/HTTP/1.0/spec>, viitattu 4.9.2024
- [13] <https://www.rfc-editor.org/rfc/rfc9114.txt>, viitattu 4.9.2024
- [14] Joarder, Y. A., and Carol Fung. "A Survey on the Security Issues of QUIC." *2022 6th Cyber Security in Networking Conference (CSNet)* /, IEEE, 2022, pp. 169–8, <https://doi.org/10.1109/CSNet56116.2022.9955622>.
- [15] <https://www.cloudwards.net/what-is-webdav/>, viitattu 4.9.2024
- [16] <https://datatracker.ietf.org/doc/html/rfc4918>, viitattu 4.9.2024
- [17] <http://www.webdav.org/specs/rfc3744.html>, viitattu 4.9.2024
- [18] <https://easy-software.com/en/glossary/webdav-protocol/>, viitattu 22.9.2024
- [19] Lombardi, Andrew. *WebSocket*. First edition., O'Reilly, 2015.
- [20] <https://datatracker.ietf.org/doc/html/rfc6455>, viitattu 5.9.2024

- [21] Ghani, Rana, et al. "Blockchain-Based E-Government System Using WebSocket Protocol." *Engineering and Technology Journal*, vol. 42, no. 4, 2024, pp. 421–29, <https://doi.org/10.30684/etj.2024.146559.1689>.
- [22] Y. Fu and M. García-Valls, "Security aspects of full-duplex web interactions and WebSockets," 2023 20th ACS/IEEE International Conference on Computer Systems and Applications (AICCSA), Giza, Egypt, 2023, pp. 1-8, doi: 10.1109/AICCSA59173.2023.10479302.
- [23] <https://http.dev/ws>, viitattu 5.9.2024
- [24] <https://tools.ietf.org/html/rfc959>, viitattu 5.9.2024
- [25] <https://www.geeksforgeeks.org/file-transfer-protocol-ftp-in-application-layer/>, viitattu 22.9.2024
- [26] <https://www.rfc-editor.org/rfc/rfc4217.txt>, viitattu 5.9.2024
- [27] <https://www.ssh.com/academy/ssh/sftp-ssh-file-transfer-protocol>, viitattu 5.9.2024
- [28] <https://datatracker.ietf.org/doc/html/draft-ietf-secsh-filexfer-02>, viitattu 5.9.2024
- [29] <https://www.thesecuritybuddy.com/network-fundamentals/what-is-an-ethernet-frame/>, viitattu 22.9.2024
- [30] https://www.ibm.com/docs/en/ssw_ibm_i_71/sqlp/rbafy.pdf, viitattu 5.9.2024
- [31] <https://www.postgresql.org/docs/current/routine-vacuuming.html>, viitattu 5.9.2024
- [32] <https://medium.com/@bindubc/distributes-system-postgres-database-system-architecture-glimse-efd7484b46b3>, viitattu 22.9.2024
- [33] <https://en.wikibooks.org/wiki/PostgreSQL/Architecture>, viitattu 5.9.2024
- [34] <https://www.mongodb.com/resources/basics/databases/document-databases>, viitattu 6.9.2024
- [35] <https://www.mongodb.com/resources/basics/databases/types>, viitattu 6.9.2024
- [36] <https://www.mongodb.com/resources/basics/databases/nosql-explained>, viitattu 6.9.2024
- [37] <https://mariadb.com/resources/blog/new-in-mariadb-xpand-6-0-columnar-indexes-for-distributed-sql/>, viitattu 7.10.2024
- [38] <https://www.ibm.com/topics/object-storage>, viitattu 6.9.2024
- [39] <https://www.redbooks.ibm.com/redpieces/pdfs/redp5537.pdf>, viitattu 6.9.2024
- [40] <https://www.snia.org/education/what-is-scsi>, viitattu 6.9.2024
- [41] <https://datatracker.ietf.org/doc/html/rfc3347>, viitattu 6.9.2024
- [42] <https://datatracker.ietf.org/doc/html/rfc4172>, viitattu 6.9.2024

- [43] <https://datatracker.ietf.org/doc/html/rfc4707>, viitattu 7.9.2024
- [44] <https://slideplayer.com/slide/6062791/>, viitattu 7.9.2024
- [45] Preston, W. Curtis, Using SANs and NAS : help for storage administrators, Sebastopol, California : O'Reilly & Associates, 2002,
- [46] <https://apix-drive.com/en/blog/reviews/microsoft-azure-review>, viitattu 7.9.2024
- [47] <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/get-started/what-is-azure>, viitattu 7.9.2024
- [48] <https://learn.microsoft.com/en-us/azure/reliability/availability-zones-overview?tabs=azure-cli>, viitattu 22.9.2024
- [49] <https://learn.microsoft.com/en-us/azure/storage/common/storage-redundancy>, viitattu 22.9.2024
- [50] <https://www.simplilearn.com/tutorials/cryptography-tutorial/rsa-algorithm>, viitattu 7.9.2024
- [51] Shamsheer Ullah, Jiangbin Zheng, Nizamud Din, Muhammad Tanveer Hussain, Farhan Ullah, Mahwish Yousaf, Elliptic Curve Cryptography; Applications, challenges, recent advances, and future trends: A comprehensive survey, Computer Science Review, Volume 47, 2023
- [52] Avestro, Junnel E., et al. "Hybrid Algorithm Combining Modified Diffie Hellman and RSA." *2019 IEEE 4th International Conference on Technology, Informatics, Management, Engineering & Environment (TIME-E)* /, IEEE, 2019, pp. 1–104, <https://doi.org/10.1109/TIME-E47986.2019.9353292>.
- [53] <https://kinsta.com/knowledgebase/tls-vs-ssl/>, viitattu 7.9.2024
- [54] <https://aws.amazon.com/what-is/ssl-certificate/>, viitattu 7.9.2024
- [55] <https://www.ssl.com/article/ssl-tls-self-signed-certificates/>, viitattu 7.9.2024
- [56] <https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/>, viitattu 7.9.2024
- [57] Käsper, E. (2012). Fast Elliptic Curve Cryptography in OpenSSL. In: Danezis, G., Dietrich, S., Sako, K. (eds) Financial Cryptography and Data Security. FC 2011. Lecture Notes in Computer Science, vol 7126. Springer, Berlin, Heidelberg. https://doi-org.libproxy.tuni.fi/10.1007/978-3-642-29889-9_4
- [58] Taparia, Ankit, et al. "Secure Key Exchange Using Enhanced Diffie-Hellman Protocol Based on String Comparison." *Proceedings of the 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)* : 22-24 March 2017, Chennai, India /, vol. 2018-January, Institute of Electrical and Electronics Engineers, 2017, pp. 174–726, <https://doi.org/10.1109/WiSPNET.2017.8299856>.
- [59] <https://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/gcm/gcm-spec.pdf>, viitattu 7.9.2024
- [60] <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.nrf52832.ps.v1.1%2Fccm.html>, viitattu 7.9.2024

- [61] <https://datatracker.ietf.org/doc/html/rfc8446>, viitattu 7.9.2024
- [62] <https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf>, viitattu 7.9.2024
- [63] Eduardo Mobilon, Dalton Soares Arantes, 100 Gbit/s AES-GCM Cryptography Engine for Optical Transport Network Systems: Architecture, Design and 40 nm Silicon Prototyping, *Microelectronics Journal*, Volume 116, 2021, 105229, ISSN 1879-2391, <https://doi.org/10.1016/j.mejo.2021.105229>.
- [64] <https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report>, viitattu 7.9.2024
- [65] <https://azure.microsoft.com/en-us/pricing/details/cosmos-db/autoscale-provisioned/>, viitattu 7.9.2024
- [66] <https://learn.microsoft.com/en-us/azure/cosmos-db/request-units>, viitattu 7.9.2024
- [67] <https://learn.microsoft.com/en-us/azure/azure-sql/database/dtu-benchmark?view=azuresql>, viitattu 7.9.2024
- [68] <https://learn.microsoft.com/en-us/azure/azure-sql/database/resource-limits-dtu-single-databases?view=azuresql>, viitattu 7.9.2024
- [69] <https://azure.microsoft.com/en-us/pricing/details/azure-sql-database/single/>, viitattu 7.9.2024
- [70] <https://azure.microsoft.com/en-us/products/storage/blobs/>, viitattu 7.9.2024
- [71] <https://azure.microsoft.com/en-us/pricing/details/storage/blobs/>, viitattu 7.9.2024
- [72] <https://azure.microsoft.com/en-us/pricing/details/storage/files/>, viitattu 7.9.2024
- [73] <https://azure.microsoft.com/en-us/pricing/details/managed-disks/>
<https://learn.microsoft.com/en-us/azure/virtual-machines/disk-bursting>, viitattu 7.9.2024
- [74] Kirilov, N., Bischoff, E. Networking Aspects of the Electronic Health Records: Hypertext Transfer Protocol Version 2 (HTTP/2) vs HTTP/3. *J Med Syst* 48, 61 (2024). <https://doi-org.libproxy.tuni.fi/10.1007/s10916-024-02080-0>
- [75] Kaur, A., Chauhan, A. P. S., & Aggarwal, A. K. (2022). Dynamic Deep Genomics Sequence Encoder for Managed File Transfer. *IETE Journal of Research*, 69(11), 7907–7919. <https://doi-org.libproxy.tuni.fi/10.1080/03772063.2022.2060869>
- [76] Łasocha, Wojciech, and Marcin Badurowicz. "Comparison of WebSocket and HTTP Protocol Performance." *Journal of Computer Sciences Institute*, vol. 19, 2021, pp. 67–74, <https://doi.org/10.35784/jcsi.2452>.
- [77] <https://candytech.in/nvme-vs-sata-vs-hdd-performance-comparison-benchmarks/>, viitattu 7.10.2024
- [78] Dominik Durner, Viktor Leis, and Thomas Neumann. 2023. Exploiting Cloud Object Storage for High-Performance Analytics. *Proc. VLDB Endow.* 16, 11 (July 2023), 2769–2782. <https://doi-org.libproxy.tuni.fi/10.14778/3611479.3611486>

- [79] <https://www.ibm.com/think/topics/man-in-the-middle>, viitattu 7.9.2024
- [80] <https://gcore.com/learning/http-vs-https-security-comparison/>, viitattu 7.9.2024
- [81] <https://www.acunetix.com/blog/articles/breach-attack/>, viitattu 7.9.2024
- [82] <https://beaglesecurity.com/blog/vulnerability/the-rc4-algorithm-in-transport-layer-security-and-secure-sockets-layer.html>, viitattu 7.9.2024
- [83] <https://www.acunetix.com/blog/articles/tls-vulnerabilities-attacks-final-part/>, viitattu 7.9.2024
- [84] <https://www.acunetix.com/blog/web-security-zone/what-is-poodle-attack/>, viitattu 7.9.2024
- [85] <https://www.acunetix.com/blog/articles/poodle-gives-final-bite-puts-ssl3-rest/>, viitattu 7.9.2024
- [86] <https://drownattack.com/>, viitattu 7.9.2024
- [87] <https://www.acunetix.com/white-papers/acunetix-web-application-vulnerability-report-2020/>, viitattu 7.9.2024
- [88] <https://www.f5.com/labs/articles/threat-intelligence/the-2021-tls-telemetry-report>, viitattu 7.9.2024
- [89] <https://www.spiceworks.com/tech/networking/articles/sftp-vs-ftp/>, viitattu 7.9.2024
- [90] <https://learn.microsoft.com/en-us/azure/role-based-access-control/scope-overview>, viitattu 7.9.2024
- [91] <https://learn.microsoft.com/en-us/azure/role-based-access-control/role-assignments-portal>, viitattu 7.9.2024
- [92] <https://learn.microsoft.com/en-us/azure/role-based-access-control/role-assignments-steps>, viitattu 7.9.2024
- [93] <https://learn.microsoft.com/en-us/entra/identity/managed-identities-azure-resources/overview>, viitattu 7.9.2024
- [94] <https://learn.microsoft.com/en-us/azure/role-based-access-control/scope-overview>, viitattu 7.9.2024
- [95] <https://learn.microsoft.com/en-us/entra/identity-platform/howto-create-service-principal-portal>, viitattu 7.9.2024
- [96] <https://www.microsoft.com/en-us/security/business/identity-access/microsoft-entra-id>, viitattu 7.9.2024
- [97] <https://learn.microsoft.com/en-us/azure/storage/common/storage-account-keys-manage?tabs=azure-portal#regenerate-access-keys>, viitattu 7.9.2024
- [98] <https://learn.microsoft.com/en-us/azure/cosmos-db/secure-access-to-data?tabs=using-primary-key>, viitattu 7.9.2024

- [99] <https://learn.microsoft.com/fi-fi/azure/storage/common/storage-account-keys-manage?tabs=azure-portal>, viitattu 7.9.2024
- [100] <https://www.wiz.io/blog/azure-active-directory-bing-misconfiguration>, viitattu 7.9.2024
- [101] <https://securityaffairs.com/124510/hacking/chaosdb-flaw-technical-details.html>, viitattu 7.9.2024
- [102] <https://chaosdb.wiz.io/>, viitattu 7.9.2024
- [103] <https://learn.microsoft.com/en-us/azure/defender-for-cloud/defender-for-storage-introduction>, viitattu 7.9.2024
- [104] <https://go.snyk.io/rs/677-THP-415/images/cloud-security-report-22.pdf>, viitattu 7.9.2024
- [105] <https://www.verizon.com/business/resources/T4d/reports/2024-dbir-data-breach-investigations-report.pdf>, viitattu 7.9.2024

